

---

# Aplicación de Clean Architecture a la gestión de la logística de exámenes en la Universidad

---



TRABAJO DE FIN DEL GRADO DEL  
INGENIERÍA DE COMPUTADORES.  
FACULTAD DE INFORMÁTICA

Lorena Costa López

Universidad Complutense de Madrid

Febrero 2021

Documento maquetado con T<sub>E</sub>X<sup>I</sup>S v.1.0.

Este documento está preparado para ser imprimido a doble cara.

# Aplicación de Clean Architecture a la gestión de la logística de exámenes en la Universidad

Memoria que presenta para optar al título de Ingeniería en Informática de  
Computadores

*Dirigida por el profesor*

**Simon Pickin**

**Universidad Complutense de Madrid**

**Febrero 2021**

Copyright © Lorena Costa López

*A mi comapañero de vida.*  
*A mi matemática favorita.*



# Resumen

En la actualidad, las facultades de la Universidad Complutense de Madrid tienen una necesidad vital de actualizar la gestión de exámenes. Este proyecto subsana todas las necesidades creando un sistema de gestión de exámenes digitalizado evitando la gran carga que supone a los profesores una organización manual. Los puntos más importantes solventados son:

- Organización y optimización del calendario de exámenes para reducir su franja de tiempo.
- Gestionar las reservas de aulas dando cabida al mayor número de exámenes por día.
- Ubicación estratégica de alumnos en aulas para evitar la copia entre ellos.
- Imprimir y almacenar bajo estimación los exámenes.
- Transporte de exámenes con sus correspondientes riesgos.

Todas estas funcionalidades son solo una pequeña muestra de las necesidades encontradas inicialmente. Todo evoluciona con el tiempo y se tiene que adaptar, los proyectos fracasan porque se hacen ad hoc[43] a unas especificaciones iniciales. Esto hace que a largo plazo queden obsoletos e inmantenibles. El potencial de este proyecto no es solo que cubre las necesidades antes nombradas sino el *know-how*<sup>1</sup> que asegura una alta progresión a futuro. Para lograr esta versatilidad se ha seguido los principios SOLID, creando una arquitectura robusta que podrá soportar cualquier evolución. Además, una gran ventaja competitiva es que la base del proyecto puede reutilizarse como estructura para otros proyectos con un fin distinto al actual.

La arquitectura del proyecto esta influenciada por el enfoque de Clean Architecture de Robert C. Martin[30] evolucionada con Hexagonal Architecture de Alistair Cockburn[31][35], que se basan en Domain-Driven Design[45]. Todo ello obliga a tener un desacople entre capas por responsabilidades y mantener una alta cohesión dentro de ellas.

Las capas utilizadas, de la parte más core de la aplicación a la más externa, son Enterprise Business Rules, Application Business Rules, Interface adapters y Frameworks & Drivers. Las capas de negocio Enterprise Business Rules y Application Business Rules definen todas la lógica de negocio como las entidades y sus caso de uso. En este dominio

---

<sup>1</sup>conjunto de conocimientos técnicos y habilidades que has adquirido a título personal

se debe definir y desarrollar toda la funcionalidad de negocio tales como los algoritmos que planificarán el calendario de exámenes, optimización de uso de aulas y distribución inteligente del alumnado en ellas. A su vez definirá, mediante contratos de interfaz, cómo han de comunicarse los módulos externos con el núcleo y que deben de implementar, lo que en Hexagonal Architecture se llama puertos. Estos agentes externos pueden ser la persistencia de datos (BBDD), la comunicación HTTP y servicios externos como la validación del alumno mediante el carnet universitario. Estas implementaciones o adaptaciones de puerto se realizarán en la capa Interface adapters, que servirá de mediador entre nuestro dominio de negocio y todo lo ajeno a él.

Uno de los focos más importantes implementados en el proyecto ha tratado la independencia con el lugar de persistencia de datos, es decir, la base de datos. Como dijo Robert C. Martin "The database is a detail. Why do we have databases?"[9] explicando su evolución desde las tarjetas perforadas hasta la posibilidad de la desaparición de las base de datos relacionales[4][11] en pro de hashtables o trees debido a la creciente velocidad de lectura de los hardware de persistencias. Esto se ha logrado mediante un simple DAO[44] en la capa de dominio y el uso de un ORM en las capas Interface adapters y Frameworks & Drivers, soportando PostgreSQL, MySQL, MariaDB, SQLite y MSSQL solo cambiando un único parámetro de código.

La lógica de negocio implementada puede ser consumida desde una página web, una aplicación móvil nativa o una aplicación de escritorio, satisfaciendo las futuras necesidades de dispositivos donde haya de ser usada. Esto aporta una gran escalabilidad y una gran proyección de futuro para añadir funcionalidades a demanda sin cambiar el core de la aplicación. Esta tenaz independencia, sumada a los principios SOLID, realizada de forma rigurosa permitirá que haya un continuo crecimiento y una fácil evolución según necesidades. Sin olvidar que presenta un precedente a seguir para futuros proyectos pudiendo seguir su estudiada arquitectura para otros fines.



# Abstract

At present, the faculties of the Complutense University of Madrid have a vital need to update the management of exams. This project solves all the needs by creating a digitized exam management system avoiding the great burden that manual organization implies for teachers. The most important points resolved are:

- Organization and optimization of the exam calendar to reduce your time frame.
- Manage classroom reservations accommodating the largest number of exams per day.
- Strategic placement of students in classrooms to avoid copying between them.
- Print and store exams under estimate.
- Transporte de exámenes con sus correspondientes riesgos.

All these features are just a small sample of the needs initially found. Everything evolves over time and it has to be adapted, projects fail because they are made ad hoc[43] to initial specifications. This makes them obsolete and unsustainable in the long term. The potential of this project is not only that it meets the needs mentioned above, but also the *know-how*<sup>2</sup> that ensures high progression in the future. To achieve this versatility, SOLID principles have been followed, creating a robust architecture that can withstand any evolution. In addition, a great competitive advantage is that the project base can be reused as a structure for other projects with a different purpose than the current one.

The architecture of the project is influenced by the Clean Architecture approach by Robert C. Martin [30] evolved with Hexagonal Architecture by Alistair Cockburn[31][35], which are based on Domain-Driven Design[45]. All this forces to have a decoupling between layers due to responsibilities and to maintain high cohesion within them.

The layers used, from the most core part of the application to the most external, are Enterprise Business Rules, Application Business Rules, Interface adapters and Frameworks & Drivers. The Enterprise Business Rules and Application Business Rules business layers define all business logic such as entities and their use cases. In this domain, all business functionality must be defined and developed, such as the algorithms that will plan the exam calendar, optimization of the use of classrooms and intelligent distribution of students

---

<sup>2</sup>set of technical knowledge and skills that you have acquired on a personal basis

in them. At the same time, it will define, by means of interface contracts, how the external modules must communicate with the kernel and what they must implement, what in Hexagonal Architecture is called ports. These external agents can be data persistence (BBDD), HTTP communication and external services such as the validation of the student through the university card. These implementations or port adaptations will be carried out in the Interface adapters layer, which will act as a mediator between our business domain and everything outside it.

One of the most important focuses implemented in the project has dealt with independence with the place of data persistence, that is, the database. As Robert C. Martin said "The database is a detail. Why do we have databases?"[9] explaining its evolution from punch cards to the possibility of the disappearance of relational databases[4][11] in favor of hashtables or trees due to increasing read speed of persistence hardware. This has been achieved through a simple DAO [44] in the domain layer and the use of an ORM in the Interface adapters and Frameworks & Drivers layers, supporting PostgreSQL, MySQL, MariaDB, SQLite and MSSQL only by changing a single parameter of code.

The implemented business logic can be consumed from a web page, a native mobile application or a desktop application, satisfying the future needs of the devices where it is to be used. This provides great scalability and a great future projection to add functionalities on demand without changing the core of the application. This tenacious independence, added to the principles of open / closed and Liskov substitution, carried out in a rigorous way, will allow for continuous growth and easy evolution according to needs. Without forgetting that it presents a precedent to follow for future projects, being able to continue its studied architecture for other purposes.

The implemented business logic can be consumed from a web page, a native mobile application or a desktop application, satisfying the future needs of the devices where it is to be used. This provides great scalability and a great future projection to add functionalities on demand without changing the core of the application. This tenacious independence, added to the SOLID principles, carried out in a rigorous way, will allow for continuous growth and easy evolution according to needs. Without forgetting that it presents a precedent to follow for future projects, being able to follow its studied architecture for other purposes.

# Repositorio

Es en este repositorio de gitHub donde se ha alojado la implementación del proyecto llamado Geloex:

`https://github.com/lorenacostace/Geloex`



# Índice

<b>Resumen</b>	<b>VII</b>
<b>Abstract</b>	<b>IX</b>
<b>Repositorio</b>	<b>XI</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Perspectiva histórica de la evaluación universitaria . . . . .	1
1.2. Situación actual en la Facultad de Informática . . . . .	1
1.3. Motivación y alcance del proyecto . . . . .	2
1.4. Covid . . . . .	3
1.5. Estructura del trabajo . . . . .	4
<b>2. Introduction</b>	<b>7</b>
2.1. Historical perspective . . . . .	7
2.2. Current situation . . . . .	7
2.3. Project scope and motivation . . . . .	8
2.4. Covid . . . . .	9
2.5. Project structure . . . . .	10
<b>3. Antecedentes</b>	<b>13</b>
3.1. Introducción . . . . .	13
3.2. Gestión de exámenes por la UNED . . . . .	13
3.3. Gestión de los exámenes en países de nuestro entorno . . . . .	15
<b>4. Análisis y especificación</b>	<b>17</b>
4.1. Descripción . . . . .	17
4.2. Enumeración . . . . .	20
<b>5. Histórico de la aplicación</b>	<b>23</b>
<b>6. Elección de las tecnologías</b>	<b>27</b>
6.1. Javascript . . . . .	28
6.2. Node.js . . . . .	28
6.3. Express . . . . .	29

6.4. Express-validation . . . . .	31
6.5. MySQL . . . . .	32
6.6. Sequelize . . . . .	32
6.7. Webstorm . . . . .	33
<b>7. Arquitectura</b>	<b>35</b>
7.1. Clean Architecture . . . . .	35
7.2. Por qué usar Clean Architecture . . . . .	37
<b>8. Instalación</b>	<b>39</b>
8.1. Introducción . . . . .	39
8.2. Webstorm . . . . .	39
8.3. Instalación Node.js . . . . .	41
8.4. Instalación npm . . . . .	42
8.5. Instalación y creación del proyecto mediante Express . . . . .	43
8.6. Refactorización del scaffolding . . . . .	43
8.7. Instalación y conexión MySQL . . . . .	46
<b>9. Implementación</b>	<b>47</b>
9.1. Introducción . . . . .	47
9.2. Implementación de la especificación . . . . .	47
9.3. Enterprise_business_rules . . . . .	48
9.3.1. Entities . . . . .	50
9.3.2. repositories . . . . .	54
9.3.3. Gestión de errores . . . . .	55
9.4. Application_business_rules . . . . .	58
9.5. Interface_adapters . . . . .	61
9.5.1. Controllers . . . . .	61
9.5.2. Routes . . . . .	64
9.6. Frameworks_drivers . . . . .	66
9.6.1. Database . . . . .	66
9.6.2. ORM . . . . .	68
9.6.3. Storage . . . . .	70
9.6.4. WebServer . . . . .	73
9.7. Escenario de uso . . . . .	74
<b>10.Verificación y Validación</b>	<b>91</b>
10.1. Introducción . . . . .	91
10.2. ESLint . . . . .	92
10.3. Given-When-Then . . . . .	93
10.4. Test unitarios . . . . .	93
10.5. Test unitarios . . . . .	94
<b>11.Trabajo futuro</b>	<b>97</b>

<b>12.Conclusiones</b>	<b>101</b>
<b>13.Conclusion</b>	<b>103</b>
<b>Bibliografía</b>	<b>105</b>





# Capítulo 1

## Introducción

### 1.1. Perspectiva histórica de la evaluación universitaria

Tras las investigaciones llevadas a cabo por los alumnos del TFG[34] de hace dos años, descubrieron que las primeras Universidades surgieron en Europa alrededor de los siglos XII y XIII a través de las escuelas catedralicias y monásticas. En ese entonces, el método de evaluación más extendido era la “disputatio”, que consistía en debatir sobre varias materias para descubrir y establecer las verdades de ellas. Había unas reglas fijas para ese proceso, tales como la dependencia de las autoridades escritas adicionales o la comprensión profunda de los argumentos de cada bando.

Con el paso del tiempo la metodología fue evolucionando hasta convertirse lo que ahora conocemos como examen oral. Este tipo de examen fue adoptado por las universidades europeas en el siglo XVIII y supuso un gran avance ya que se considera el primer método ecuanime racional de evaluación de estudiantes.

En los últimos años ha habido un gran avance tecnológico en muchos ámbitos diferentes. Pese a ello, la gestión logística de los exámenes en las universidades casi no ha avanzado, siendo en la mayor parte de los casos un método muy ineficiente y sin automatizar.

### 1.2. Situación actual en la Facultad de Informática

En el TFG[34] de los alumnos de hace dos años explican que, a día de hoy, la Facultad de Informática, perteneciente a la Universidad Complutense de Madrid, no dispone de un gran avance en la logística de sus exámenes. El hecho de no tener un sistema automatizado implica un trabajo extra que recae en el personal docente. Gracias al progreso tecnológico, se podría liberar a los profesores de ese trabajo adicional.

El proceso actual de los exámenes en la Facultad de Informática es:

1. Publicación del calendario de exámenes a principio del año académico.

## 2. Asignación de las aulas para el examen.

- Reserva del aula para la parte teórica (si existe) en función del total de alumnos matriculados.
- Reserva de laboratorios estimando al alza el número de alumnos que van a presentarse para la parte práctica (si existe).

## 3. Impresión de los exámenes estimando al alza el número de alumnos que van a presentarse.

## 4. Transporte de las copias del examen al aula correspondiente el día del examen.

No solo el hecho de estimar las copias necesarias del examen puede llegar a resultar tedioso, sino también transportar las mismas al aula, con el consecuente riesgo de pérdida en el proceso desde su impresión hasta el día del examen.

La metodología utilizada puede provocar distintos contratiempos y problemas como pueden ser:

- Trabajo invertido por parte de los técnicos en la preparación del laboratorio, y que finalmente no sea necesario por presentarse menos alumnos, llegando incluso a quedar laboratorios completamente vacíos en ocasiones.
- Posibles exámenes impresos sobrantes o aulas que finalmente han quedado vacías con el posible gasto en calefacción o aire acondicionado necesitado para crear un buen clima para el examen. Además de lo mencionado, gracias a este proyecto, sería posible reducir el programa de exámenes, debido a la optimización de aulas y además la consiguiente reducción del profesorado de vigilancia en exámenes.
- La pérdida de tiempo del profesor en estimar el número de alumnos que podría presentarse a cada examen, lo que ello conlleva la consiguiente pérdida de tiempo en reservar cada una de las aulas para cada uno de sus exámenes.
- La colocación de alumnos en exámenes. El profesor debe invertir tiempo en ordenar a los alumnos, con el fin de que la separación entre ellos sea la propicia para evitar posibles trampas.
- Problema de que los alumnos copien. Si un alumno es libre de sentarse en cualquier sitio, las probabilidades de copiar cambiarán en función del sitio escogido. Si puede sentarse al lado o muy cerca de algún amigo, el riesgo de copia es mayor. En cambio, si es consciente de que los sitios son asignados, sabe que sus posibilidades de copiar son menores ya que igual no puede sentarse cerca de un amigo, en el caso de que lo necesite para aprobar el examen.

## 1.3. Motivación y alcance del proyecto

En el punto anterior, se ha descrito la situación actual de la metodología de los exámenes y los posibles inconvenientes tales como la pérdida de tiempo que puede suponer

a un profesor la gestión de reservar un aula o la colocación de los alumnos en el aula destinada al examen. Pero también puede existir otros problemas como puede ser la copia durante un examen, o incluso la pérdida del mismo.

Este TFG pretende paliar, en la medida de lo posible, los inconvenientes y problemas que puedan surgir en lo referente a la logística de exámenes. Para ello, mis compañeros[34] de hace dos años diseñaron una aplicación que permite la gestión de reserva de aulas y la asignación de asientos para los alumnos en base a un algoritmo. Esto permite prevenir la copia en exámenes, dado que los asientos serán asignados por el algoritmo en función de unos parámetros previos.

La aplicación también gestionará un examen en general, pudiéndose crear un examen, mostrar toda la información referente a él (asignatura, grupo, fecha, aula, alumnos, profesor...), modificarlo o incluso eliminarlo. Además, el profesor responsable tendrá la posibilidad de subir el pdf con el modelo de examen en la aplicación, asignándolo a un examen creado previamente. Por otra parte, los exámenes únicamente podrán ser creados por un usuario que posea el rol de administrador de exámenes. Otra de las opciones que ofrecerá la aplicación será la subida de la solución de un alumno al examen, pudiendo ser ésta el escaneo del examen escrita por el alumno durante el examen, o en caso de ser un examen de programación, subir los ficheros requeridos por el profesor. Esta opción estará sujeta al inicio y final de un examen, ya que será imposible añadir una solución de examen a un examen que no haya comenzado o haya finalizado, lo cual, será dirigido por el profesor que ha estado impartiendo la asignatura o por el profesor de guardia asignado a la vigilancia de dicho examen.

Para la gestión de lo explicado anteriormente, será necesario disponer de la información de los alumnos matriculados y las asignaturas. Otro requisito importante es el conocimiento del calendario de exámenes, la definición automática de un calendario de exámenes que satisfaga una serie de requisitos es un problema de optimización compleja que no se trata en este TFG; suponemos que el calendario viene definida. Esto es importante dado que la idea principal es optimizar las aulas y asientos para cada uno de los exámenes y es probable que en un mismo día se realicen varios exámenes a la vez. Por último, será necesario conocer en detalle cada aula, es decir, el número de asientos y su distribución.

Se pretende crear una aplicación que permita mejorar el funcionamiento de nuestra facultad, y que pueda ser ampliable a otras facultades. A continuación, se presentará la aplicación, explicando los lenguajes seleccionados y su porqué, las tecnologías utilizadas en el proceso de desarrollo, el funcionamiento, los requisitos y la arquitectura elegida.

## 1.4. Covid

Dada la situación actual de pandemia que se está viviendo, este proyecto se ha convertido en un trabajo indispensable. Inicialmente las necesidades no venían a reflejar los tiempos propicios, y por tanto las funcionalidades no tenía el foco adecuado, a pesar de ello pudimos ver los beneficios que daría la aplicación.

Uno de los principios para combatir una pandemia una vez controlada, es evitar aglomeraciones que es justo lo que ocurre en época de exámenes. A lo largo de los pasillos de la facultades, los alumnos se agolpan alrededor de las puertas de las aulas para obtener los asientos más cómodos para ellos o ser los primeros en entrar. La solución ya se encontraba en las funcionalidades recogidas, concretamente con la asignación de plazas en las aulas. Dentro de esta característica tenemos dos variantes:

- Modo estático: Los alumnos conocen de antemano el asiento que tienen asignado en la aula antes de llegar al examen mediante una preinscripción. Incluso se podrían organizar la cola del examen en función de tu asiento, rellenando la clase de fondo a frente.
- Modo dinámico: Según llega el alumno al aula y se identifica, automáticamente se le asigna un asiento.

Aunque es difícil controlar el acceso al aula para que se haga escalonado, si que se puede tener un control de la salida en cierto modo. Cuando un alumno recogiera el examen de la impresora empezaría a contar su tiempo para desarrollarlo. De esta forma la salida se produciría ligeramente más escalonada dejando margen de mejora a futuro. Entre las mejoras que se podrían contar, cabría la posibilidad de que cuando alumno solicite terminar el examen se le indique en que momento debe abandonar el aula. Las indicaciones pueden llegar mediante una notificación a una aplicación móvil.

La otra gran medida para rastrear casos positivos es tener una buena trazabilidad de las personas para identificar contactos con gente positiva a la enfermedad. Aunque existen apps que ayudan a dicho fin, dentro de las aulas, la aplicación desarrollada complementaria esa finalidad. Se podría saber la ubicación exacta del alumno que ha dado positivo en el aula y todos los alumnos que se sentaron alrededor, pudiendo de esta forma trazar círculos de cuarentena donde los alumnos más cercanos sentados a él serían contactos de gravedad alta y se notificaría en consecuencia.

## 1.5. Estructura del trabajo

Tras la Introducción, dada por la perspectiva histórica de la evaluación universitaria, la situación actual en la Facultad de Informática y la Motivación y alcance del proyecto, y Covid, se va a exponer a continuación un breve resumen del contenido del resto de la memoria, comprendiendo estos los siguientes capítulos:

- El siguiente Capítulo es el 2, Introduction. Siendo éste la traducción al inglés del Capítulo 1, Introducción.
- El próximo capítulo es el 3, Antecedentes. En este capítulo se va a exponer cómo otras universidades gestionan los exámenes y el método utilizado para ello.

- Después le sigue el capítulo 4, Análisis y especificación, en el cual se realiza una descripción del proyecto y dos especificaciones en función de un requisito previo.
- En el Capítulo 5 es el correspondiente al histórico de la aplicación, mostrando la evolución de la aplicación desde que empezó hace dos años, hasta ahora.
- El capítulo 6 continúa con la elección de las tecnologías, en el que se exponen los motivos de la elección de cada una de ellas y sus ventajas principales.
- El capítulo 7 es el correspondiente al de Arquitectura, explicando la arquitectura escogida y los motivos de su elección.
- Es en el capítulo 8 en el que se enseña la instalación, organización e implementación que se ha realizado en el proyecto, así como un ejemplo de un escenario de uso para mayor comprensión.
- En el Capítulo 9, Implementación, es en el se explicará como se ha implementado cada parte de la aplicación y se expone un ejemplo de un escenario de uso.
- El capítulo 10 es el correspondiente a la verificación y validación. En éste capítulo se explica las validaciones y verificaciones necesarias para el proyecto en función del nivel de progreso que tenga.
- A continuación, se encuentra el capítulo 11, en el que se realiza un desarrollo detallado del trabajo futuro haciendo incapié en mantener la escalabilidad del proyecto y su solidez.
- El Capítulo 12 es Conclusiones. En este capítulo, se realiza un resumen de las partes más relevantes de la aplicación, además de una pequeña parte de los conocimientos adquiridos.
- El Capítulo 13 es la traducción al inglés del Capítulo 12, Conclusiones.
- Por último, se encuentra la bibliografía, en el que se encuentran las páginas y documentos que se han utilizado tanto para aprendizaje, para instalación, como para contraste de tecnologías entre otras.



# Capítulo 2

## Introduction

### 2.1. Historical perspective

After the investigations carried out by the students of the TFG [34] two years ago, they discovered that the first Universities arose in Europe around the 12th and 13th centuries through the cathedral and monastic schools. At that time, the most widespread evaluation method was the “disputatio”, which consisted of debating on various subjects to discover and establish the truths of them. There were set rules for that process, such as reliance on additional written authorities or deep understanding of each side’s arguments.

Over time the methodology evolved to become what we now know as the oral exam. This type of examination was adopted by European universities in the 18th century and represented a great advance since it is considered the first equanimous rational method of evaluating students.

In recent years there has been a great technological advance in many different areas. Despite this, the logistics management of the exams in the universities has hardly advanced, being in most cases a very inefficient and non-automated method.

### 2.2. Current situation

In the TFG of the students ed two years ago they explain that, to this day, the Faculty of Informatics, belonging to the Complutense University of Madrid, does not have a great advance in the logistics of its exams. The fact of not having an automated system implies an extra work that falls on the teaching staff. Thanks to technological progress, teachers could be freed from that extra work.

The current process of the exams in the Faculty of Informatics is:

1. Publication of the exam calendar at the beginning of the academic year.
2. Assignment of classrooms for the exam.

- Classroom reservation for the theoretical part (if any) depending on the total number of students enrolled.
  - Reserve laboratories estimating upwards the number of students who are going to attend the practical part (if any).
3. Printing of the exams estimating upwards the number of students who are going to sit.
  4. Transportation of the exam copies to the corresponding classroom on the exam day.

Not only the fact of estimating the necessary copies of the exam can be tedious, but also transporting them to the classroom, with the consequent risk of loss in the process from printing to exam day.

The methodology used can cause different setbacks and problems such as:

- Work invested by the technicians in the preparation of the laboratory, and that in the end it is not necessary because fewer students show up, sometimes even leaving laboratories completely empty.
- Possible leftover printed exams or classrooms that have finally been left empty with possible spending on heating or air conditioning needed to create a good exam climate. In addition to the aforementioned, thanks to this project, it would be possible to reduce the exam program, due to the optimization of classrooms and also the consequent reduction in the number of exam surveillance teachers.
- The loss of time for the teacher in estimating the number of students who could take each exam, which entails the consequent loss of time in reserving each of the classrooms for each of their exams.
- Student placement in exams. The teacher must invest time in ordering the students, so that the separation between them is conducive to avoid possible traps.
- Problem that students copy. If a student is free to sit anywhere, the odds of copying will change depending on the site chosen. If you can sit next to or very close to a friend, the risk of copying is greater. On the other hand, if you are aware that the places are assigned, you know that your chances of copying are lower since you can not sit near a friend, in case you need it to pass the exam.

## 2.3. Project scope and motivation

In the previous point, the current situation of the methodology of the examinations and the possible inconveniences such as the loss of time that a teacher can suppose when reserving a classroom or the placement of students in the classroom destined to exam. But there can also be other problems such as copying during an exam, or even losing it.

This TFG aims to alleviate, as far as possible, the inconveniences and problems that may arise in relation to the logistics of exams. To do this, my colleagues [34] two years



ago designed an application that allows the management of classroom reservations and the allocation of seats for students based on an algorithm. This allows to prevent copying in exams, since the seats will be assigned by the algorithm based on previous parameters.

The application will also manage an exam in general, being able to create an exam, display all the information related to it (subject, group, date, classroom, students, teacher ...), modify it or even delete it. In addition, the responsible teacher will have the possibility of uploading the pdf with the exam model in the application, assigning it to a previously created exam. On the other hand, exams can only be created by a user who has the role of exam administrator. Another option that the application will offer will be the uploading of a student's solution to the exam, which may be the scan of the exam written by the student during the exam, or in the case of a programming exam, upload the files required by the teacher. This option will be subject to the beginning and end of an exam, since it will be impossible to add an exam solution to an exam that has not started or has finished, which will be directed by the professor who has been teaching the subject or by the professor. guard assigned to oversee said examination.

For the management of the previously explained, it will be necessary to have the information of the enrolled students and the subjects. Another important requirement is knowledge of the exam calendar. The automatic definition of an exam calendar that satisfies a series of requirements is a complex optimization problem that is not dealt with in this TFG; We assume that the calendar is defined. This is important since the main idea is to optimize the classrooms and seats for each of the exams and it is likely that several exams will be taken on the same day at the same time. Finally, it will be necessary to know each classroom in detail, that is, the number of seats and their distribution.

It is intended to create an application that allows us to improve the functioning of our school, and that can be extended to other schools. Next, the application will be presented, explaining the selected languages and why, the technologies used in the development process, the operation, the requirements and the chosen architecture.

## 2.4. Covid

Given the current pandemic situation that is being experienced, this project has become an indispensable job. Initially, the needs did not reflect the favorable times, and therefore the functionalities did not have the appropriate focus, despite this we could see the benefits that the application would give.

One of the principles to combat a pandemic once controlled, is to avoid some events, which is just what happens at exam time. Along the hallways of the faculties, students crowd around the classroom doors to get the most comfortable seats for them or to be the first to enter. The solution was already found in the functionalities collected, specifically with the allocation of places in the classrooms. Within this feature we have two variants:

- Static mode: Students know in advance the seat they have assigned in the classroom before arriving at the exam through a pre-registration. They could even organize the exam queue based on your seat, filling in the class from front to back.
- Dynamic mode: As the student arrives in the classroom and identifies himself, he is automatically assigned a seat.

Although it is difficult to control access to the classroom so that it is staggered, it is possible to control the exit in a certain way. When a student picked up the exam from the printer, they would start counting their time to develop it. In this way, the output would be slightly more staggered, leaving room for future improvement. Among the improvements that could be counted, it would be possible that when the student requests to finish the exam, he/she will be told when to leave the classroom. Directions can be reached by notification to a mobile app.

The other great measure to track positive cases is to have a good traceability of people to identify contacts with people positive to the disease. Although there are apps that help to this end, within the classrooms, the developed application complements that purpose. It would be possible to know the exact location of the student who has tested positive in the classroom and all the students who sat around it, thus being able to draw quarantine circles where the closest students seated to him would be high severity contacts and he would be notified accordingly.

## 2.5. Project structure

After the Introduction, given by the historical perspective of the university evaluation, the current situation in the Faculty of Informatics and Motivation and scope of the project, and Covid, a brief summary of the content of the rest of the report will be presented below, comprising the following chapters:

- The next Chapter is 2, Introduction. This being the English translation of Chapter 1, Introduction.
- The next chapter is 3, Background. This chapter is going to explain how other universities manage the exams and the method used to do so.
- This is followed by Chapter 4, Analysis and Specification, in which a project description and two specifications are made based on a prerequisite.
- Chapter 5 is the one corresponding to the history of the application, showing the evolution of the application since it began two years ago, until now.
- Chapter 6 continues with the choice of technologies, in which the reasons for the choice of each of them and their main advantages are exposed.
- Chapter 7 is the one corresponding to Architecture, explaining the chosen architecture and the reasons for its choice.

- 
- It is in chapter 8 that the installation, organization and implementation that has been carried out in the project is taught, as well as an example of a use scenario for greater understanding.
  - In Chapter 9, Implementation, it will be explained how each part of the application has been implemented and an example of a usage scenario is presented.
  - Chapter 10 is the one corresponding to verification and validation. This chapter explains the necessary validations and verifications for the project depending on the level of progress it has.
  - This is followed by Chapter 11, in which a detailed development of future work is carried out with an emphasis on maintaining the scalability of the project and its solidity.
  - Chapter 12 is Conclusions. In this chapter, a summary of the most relevant parts of the application is made, in addition to a small part of the knowledge acquired.
  - Chapter 13 is the English translation of Chapter 12, Conclusions.
  - Finally, there is the bibliography, which contains the pages and documents that have been used both for learning, for installation, and for contrasting technologies, among others.



# Capítulo 3

## Antecedentes

### 3.1. Introducción

Para llevar a cabo el desarrollo de la aplicación, ha resultado de gran ayuda la investigación acerca de la gestión de exámenes de otras universidades que llevaron a cabo mis compañeros[34] de hace dos años. Después del análisis en varias universidades, se observa que la que tiene el sistema más avanzado es la Universidad Nacional de Educación a Distancia (UNED). En ella hay métodos que no se pueden llevar a cabo en la Universidad Complutense de Madrid debido a la centralización de exámenes, pero hay otros que sí se pueden replicar en la Facultad de Informática.

Una vez recabada toda la información acerca de la gestión de exámenes en las distintas plataformas que dispone cada universidad, se ha llegado a la conclusión de que no existe un método unánime que se siga para la gestión de exámenes, sino que cada universidad tiene su propio método. Estos métodos pueden coincidir en ciertas características como pueden ser la identificación en un examen, pero en la mayoría de los aspectos no coinciden.

A continuación, se va a llevar a cabo una explicación sobre la información obtenida y la posible aplicación o comparativa con la Universidad Complutense de Madrid.

### 3.2. Gestión de exámenes por la UNED

Según la información que mis compañeros[34] de hace dos años llegaron a conseguir, previamente a 2010, los exámenes eran impresos en Madrid. Mediante sorteo se elegía el centro de la UNED donde participaban cada profesor en el tribunal. A dicho centro eran transportados los exámenes en valijas cerradas bajo llave. En el transcurso desde que llegaban los exámenes al centro, hasta el día del examen, un profesor perteneciente al tribunal debía vigilar dichos exámenes. Para recoger los exámenes era necesario ser presidente del tribunal e ir previamente a por la llave, con la consecuente probabilidad del extravío de la misma en el trayecto. Fue en 2010 cuando comenzaron a usarse las primeras implementaciones de “La Valija Virtual”. Esta implementación permitió realizar

los exámenes más rápido y de manera más segura, ya que no era necesario ningún tipo de transporte, tanto de las valijas con los exámenes como de la llave.

Gracias a “La Valija Virtual”, no es necesario que el profesor distribuya o redistribuya a los alumnos en un examen, ya que utiliza un sistema dinámico de asignación de asiento y aula, en caso de ser necesaria más de un aula, mediante la identificación del alumno cuando llega al examen. Cuando el alumno se identifica, se imprimen los datos de la ubicación que va a ocupar, además del enunciado del examen. Este sistema proporciona una gran seguridad en lo referente a copias directas, dado que por cercanía será imposible copiar de otro compañero.

En lo referente al profesor, dispone de una interfaz sobre el aula, mostrando un mapa de la misma, permitiendo pinchar en cada puesto con el fin de obtener más información acerca del alumno ubicado en esa posición, como puede ser el examen que está realizando, o datos informativos acerca del alumno. Además, dispone de un sistema de avisos, que informa al profesor de que el tiempo de finalización del examen de un alumno está próximo, cambiando para ello el color del puesto a rojo.

de que tiempo de finalización del examen de un alumno está próximo, mediante el cambio de color en el puesto a rojo.

El proceso comienza importando la información del calendario de exámenes. Una vez el sistema tiene este calendario, el transcurso del examen empieza con el descifrado de este mediante un miembro del Tribunal. Una vez llegado el día del examen, Para hacer efectivo el descifrado es necesario la tarjeta universitaria del miembro del Tribunal. Una vez descifrado el examen, dará comienzo la sesión del examen. Para realizar el examen, el alumno deberá acercarse a un lector de identificación, mediante DNI o tarjeta universitaria, que le reconocerá e imprimirá el examen correspondiente. En la cabecera de dicho examen aparecerá el puesto donde debe realizar el examen, el DNI, el código de la asignatura, el código de la escuela, el material que se le permite utilizar durante el examen, y la hora de finalización del examen, calculado en función de la hora de impresión del examen.

Cabe la posibilidad de que se produzcan aglomeraciones en torno al lector a la hora del examen. Para evitar esto el lector se controla a través de un sistema que simula un semáforo, indicando de este modo cuando el sistema está preparado para recibir al siguiente alumno.

A lo largo del examen, se lleva a cabo una comprobación de identificación de la persona con el puesto que ocupa. Además, se dispone de un sistema de notificaciones para consultar el tiempo de examen que dispone el alumno.

Cuando el alumno ha finalizado el examen, se procederá a su escaneo por el personal del tribunal. Al escanear el examen, se crea una copia de este, permitiendo así que el profesor pueda corregirlo en ese mismo momento y a continuación la entrega es registrada y se envía una copia cifrada a la sede. Es importante que el miembro del tribunal se asegure de que el escaneo del examen comience con la primera hoja, puesto que es de

esa hoja desde donde se leen de manera automática los datos del alumno, la asignatura y la escuela. Este sistema resulta ser una garantía ante pérdidas, dado que antes existía la posibilidad de extraviar algún examen en el trayecto hasta la sede.

El sistema también ofrece la posibilidad de obtener una lista con los exámenes pendiente de entrega, así como la de los alumnos presentados.

### 3.3. Gestión de los exámenes en países de nuestro entorno

Mis compañeros[34] del año pasado hicieron una búsqueda exhaustiva en la que se puede llegar a la conclusión de que las universidades no disponen de gran información acerca de la gestión de sus exámenes. Algunas de las universidades que ofrecían algo más de información son las siguientes:

- Technical University of Denmark (Dinamarca)
- University of Oulu (Finlandia)
- University of Helsinki (Finlandia)
- Mälmo University (Suecia)
- Tallinn University (Estonia)
- University of Bergen (Noruega)

Todas las universidades disponen de un plazo de preinscripción a cada examen. En cambio, no todas tienen las mismas consecuencias si un alumno inscrito a un examen no se presenta. En el caso de la Universidad de Tallin, si un alumno inscrito no se presenta a un examen, es marcado como ausente. Cuando es marcado como ausente, ese alumno suspenderá de manera automática dicha asignatura y además no podrá optar a la recuperación.

Al igual que disponen de un periodo de inscripción a un examen, también disponen de un periodo de cancelación. Este periodo varía en función de la Universidad. Para el caso de la Universidad de Tallin, es posible realizar la cancelación de inscripción al examen hasta un día antes de este. Otro ejemplo de esto es el de la Universidad de Helsinki, en la cual el periodo máximo de cancelación de inscripción a un examen es el de 10 días antes del examen.

Como en estas universidades, el objetivo sería incorporar un método análogo en nuestro proyecto. Este método proporciona una forma eficiente de distribución de los alumnos registrados de un examen. En función a las preinscripciones obtenidas, será posible el caso de tener más de un examen en una misma aula.

Al conocer el número exacto de alumnos presentados a un examen, resulta sencilla la asignación de puestos a cada alumno. El requisito necesario para el proceso será la identificación del alumno mediante la tarjeta universitaria. Existe la posibilidad de que, si por alguna circunstancia no justificable, un alumno inscrito decida no presentarse al examen. En este caso la convocatoria del examen correrá. Si un alumno no está inscrito en un examen, no será posible que se presente al mismo.

La identificación es obligatoria en todas las universidades. En la mayoría es obligatorio la identificación a final del examen, excepto en la de Malmö, en la cual es necesario identificarse antes de este.

Al igual que en Dinamarca, la Facultad de Informática dispone de aulas de gran capacidad. Esto resulta ser una gran ventaja dado que pueden juntarse más de un grupo o examen a la vez. El conseguir juntar grupos de la misma asignatura o exámenes de asignaturas distintas ofrece la posibilidad de acortar el periodo de exámenes y de disminuir el número de profesores de guardia que se necesitan para vigilar todos los exámenes.

En la Universidad de Oulu, la mayor parte de los exámenes que se realizan son en aulas, siendo puntuales los casos en los que utilizan los laboratorios para examinarse. Pese a ello, en los exámenes en laboratorio, cada alumno tiene un puesto preasignado. En la Universidad Complutense de Madrid no es así. Muchos de los exámenes realizados son en laboratorio, esto produce que en bastantes ocasiones se necesite reubicar a los alumnos en un mismo laboratorio, quedándose de este modo, laboratorios vacíos. Con este sistema, este problema se solventaría, y se ahorraría trabajo al personal de laboratorios ya que no sería necesario preparar para el examen más laboratorios que los que se van a usar finalmente.



# Capítulo 4

## Análisis y especificación

### 4.1. Descripción

Geloex es un proyecto que tiene como objetivo automatizar en la medida de lo posible la gestión de los exámenes.

La aplicación va a tener cuatro roles de usuario, los cuales podrán realizar tareas diferentes y es por ello que el análisis de la aplicación se va a realizar en función de esos usuarios.

Se va a comenzar con el primer rol, el administrador de sistemas. Este usuario va a ser el encargado de la gestión de otros usuarios. La gestión de usuarios implica poder crear un usuario, modificar un usuario ya creado, eliminar un usuario o mostrar la información de un usuario. El administrador de sistemas dispondrá de una interfaz en la que le salga un listado de los usuarios creados. Además de ello, dispondrá de varias pestañas para realizar filtrado, siendo éstos por nombre, ordenando los usuarios por orden alfabético y también tendrá la posibilidad de ordenación por fecha de creación de usuario. Otras opciones de filtrado que implican la reducción de muestra de usuarios serían la de filtrado por grupo, en la que le aparecerá un listado con los grupos y podrá seleccionar uno de ellos. Otra opción de filtrado será el de nombre, poniendo el nombre del usuario a filtrar mostrando únicamente los usuarios con ese nombre. Además de lo anterior, tendrá la opción de filtrar por rol, seleccionando el rol de un desplegable. Existirá también la opción del filtrado por asignatura, siendo ésta seleccionada mediante un desplegable en la que aparecerá el nombre de cada una. Y por último, se podrá filtrar por curso, seleccionando en cual de ellos se desea buscar. Éstas dos últimas opciones serán para la filtración de alumnos, pudiéndose estudiar la posibilidad de añadir a los profesores que impartan en esos grupos o en la asignatura. Cuando el administrador de sistemas seleccione un usuario, aparecerá otra vista en la que se muestre toda la información acerca del usuario seleccionado. En esta vista también será posible modificar el usuario o eliminarlo.

El segundo rol es para el administrador de exámenes será similar al de sistemas, pero en vez de con usuarios, con exámenes. En la página principal dispondrá de una lista con todos los exámenes creados. Tendrá la opción de ver en detalle un examen si lo selecciona,

además de poder modificar cualquiera de sus campos o eliminar el examen por completo. Dispondrá de varias opciones de filtrado, por asignatura, seleccionando en un desplegable la asignatura por la que se desea filtrar. También podrá filtrar por fecha de creación del examen, es decir, la fecha en la que se ha creado el examen en el sistema. Otra opción de filtrado será por profesor, mostrando el número de exámenes de los que un profesor está a cargo. Además será posible filtrar por grupo, mostrando así todos los exámenes que han sido creados para un grupo concreto. Otra opción, tal como el grupo, es la de filtración por curso, obteniendo todos los exámenes referentes al curso seleccionado. Además de ello, se podrá filtrar por estado de examen, comprendiendo éste tres posibles estados: por hacer, en progreso o finalizado. Tendrá la opción de filtrar por duración de examen. Otra opción será el filtrado mediante la fecha de examen, mostrando los exámenes que serán realizados en esa fecha. Y por último, el aula, mostrando para el aula seleccionada los exámenes que se realizarán ahí durante todo el periodo de exámenes.

Otro rol va a ser el de el alumno. Los alumnos van a disponer de dos vistas diferentes. La primera de ellas será en la que van a poder consultar todos los exámenes que pueden realizar. Al igual que los administradores, los alumnos también van a disponer de opciones de filtrado para agilizar la búsqueda de exámenes. Aunque pueden filtrar, los filtros del alumno estarán sujetos a lo que está matriculado, es decir, cuando filtre por grupo, curso, asignatura o profesor, solo optará a las asignaturas, cursos y grupos en los que está matriculado, así como los profesores que le imparten clase. También tendrá la opción de filtrar por fecha de examen. Y la última opción de filtrado de la que va a disponer es la de por estado de examen, pudiendo seleccionar un examen que todavía no se ha realizado, o un examen que ya ha finalizado. No se contempla la posibilidad de que un alumno tenga la opción de filtrar el estado de un examen como 'En progreso', ya que se le consigue ver la utilidad. La otra vista será la de laboratorio. En esta vista, podrá ver el título de la asignatura de la que se está examinando, y contar además con cinco botones. El primero de ellos será con el que solicite ayuda al profesor por tener una duda. El siguiente será el correspondiente al examen, pudiendo descargarlo para visualizarlo desde el ordenador. Otro será el de solicitud de escaneo de la solución del examen, para el caso en el que sea necesario. Y por último dispondrá de un botón en el que pueda ser posible adjuntar los archivos generados durante el examen y que sean necesarios para el profesor para llevar a cabo la corrección. Ahora se va a explicar el proceso de examen para un alumno. Una vez llegue el alumno al aula correspondiente, se autenticará mediante el carné universitario o introduciendo los caracteres de su DNI. A continuación, se le imprimirá el examen que debe realizar, además de los datos correspondientes del alumno, como el nombre, DNI, asignatura, grupo, fecha, y puesto que debe ocupar. Cuando el alumno finalice el examen, deberá desloguearse para indicar que ha finalizado el examen. Éste logout se realizará volviendo a pasar su carné universitario o introduciendo su DNI.

El último de los roles es el correspondiente al profesor. Al igual que el rol del alumno, el profesor va a disponer de dos posibles vistas. La primera será donde se le mostrarán todos los exámenes de las asignaturas que imparte. Para una búsqueda más ágil, será posible tener varias opciones de filtrado. Al igual que los alumnos, las opciones de filtrado de un profesor están sujetas a lo que imparte, teniendo para filtrar las asignaturas de las que es el profesor, el filtrado por curso en los que tiene alguna asignatura a su cargo, o el

grupo, teniendo como opción los grupos en los que imparte alguna asignatura. También tendrá la opción del filtrado por fecha de examen, además de por estado, siendo las dos opciones posibles las correspondientes a los exámenes por realizar, y los que ya han finalizado. Cuando se seleccione un examen, y el estado del examen sea **Finalizado**, será posible descargar las soluciones de los alumnos. La otra vista es la correspondiente a un examen. En la vista principal tendrá en la parte izquierda un menú, y la parte central estará compuesta por una representación del aula seleccionada. La parte del menú de la izquierda contiene un listado con el nombre de las aulas, para el caso en el que un examen requiera más de un aula, un apartado de notificaciones, en el cual aparecerá un listado de los puesto que han solicitado ayuda, y un tercer apartado de Modelos de examen. En el listado de aulas, como ya se ha anticipado, se va a mostrar un listado con el número de aulas en las que se realiza el examen. Cuando un aula es seleccionada, en la parte central de la pantalla aparece una representación gráfica del aula. Esta representación indica los puestos, pudiendo variar el color del puesto según si está vacío, si un alumno está realizando un examen, si el alumno en ese puesto ha solicitado ayuda o si está próximo a agotar el tiempo de su examen. Si un alumno está realizando el examen, el profesor tendrá la opción de pinchar en ese puesto y que en una ventana emergente, se muestra toda la información acerca de ese alumno, además en esa ventana tendrá la opción de añadir un comentario para ese alumno. Desde la vista principal dispondrá de dos botones, uno para finalizar el examen, por motivos como puede ser que todos los alumnos han entregado el examen antes de que el tiempo de duración del examen haya finalizado, o porque nadie se haya presentado al examen. Y el otro botón permitirá ampliar el tiempo del examen.

El proceso de la reserva de aulas se llevará a cabo tras la creación de todos los exámenes. Un administrador de exámenes dispondrá de un botón en el que, tras haber creado todos los exámenes, podrá pinchar y se generará el calendario de exámenes de ese curso académico. Esta generación de calendario lleva implícita la asignación de aulas, y por tanto, la reserva de las mismas. Para realizar correctamente este proceso de calendario, será necesario previamente que todos los exámenes hayan sido creados, que dispongan de tiempo de duración, que tengan el número estimado de alumnos que pueden presentarse y el número de modelos que tiene cada examen.

El proceso de la asignación de asiento a un alumno se realizará cuando con la identificación del alumno cuando entra en el aula en el momento del examen. En ese momento, se le asignará un asiento en ese aula.

Otro proceso sería el de recogida de las soluciones al examen de los alumnos. Este proceso tiene dos posibilidades, que el examen se realice escrito, se escanee al final del examen y la solución sea subida a la plataforma, o que sea un examen de laboratorio y se requiera la subida de ficheros, los cuales se podrán realizar mediante la vista del examen del alumno.

## 4.2. Enumeración

La gestión de exámenes en la Universidad, en particular, la parte de la gestión de aulas y la asignación de asientos, tiene una alta complejidad. A continuación se va a mostrar los dos posibles modos, en función de una preinscripción o no, que llegaron a exponer mis compañeros de hace dos años[34].

El modo dinámico se corresponde con la opción en la que no es necesaria una preinscripción. En este modo el sistema obtendrá la información del alumno en el momento en el que se realiza el examen, por lo que los puestos se deberán asignar en ese momento. Para que los puestos se asignen correctamente se utilizarían algoritmos de coloreado de grafos. En ocasiones, se recurrirá a la estimación de alumnos que pueden presentarse a un examen, ya sea porque en algunas asignaturas sea necesario cumplir con ciertos requisitos, como la entrega de todas las prácticas, o porque al contar con una estimación, y no con todo el número de alumnos matriculados en cada asignatura, obtendríamos un resultado más próximo a la optimización completa del aula.

Es probable que en ocasiones sea necesario utilizar algún tipo de estimación, debido a que en ciertas asignaturas es necesario cumplir ciertos requisitos para presentarse al examen, como la entrega completa de todas las prácticas, y con respecto a tener en cuenta a todos los matriculados en la asignatura.

El otro modo es el estático. Este modo se basa en una preinscripción previa al examen. Con esta preinscripción, el sistema dispone de los datos necesarios para asignar los puestos en el examen de la manera más óptima. Para la preinscripción, se proporcionará una plataforma a los alumnos en la que dispondrán de todos los exámenes con los días y horas a las que optan a presentarse. Cada examen tendrá un plazo de inscripción, por lo que, tras finalizar ese periodo, ningún alumno podrá inscribirse. Al igual que el modelo dinámico, dispondrá de una posible estimación en el caso de que sea necesario cumplir ciertos requisitos previos al examen como ya se han mencionado.

En función a la descripción de los dos modos, pueden diferenciarse varios niveles de complejidad para cada uno de ellos. Este desarrollo fue analizado por mis compañeros del curso anterior de una forma correcta y específica. A continuación, se muestra dicha enumeración[34]:

1. “Grado de complejidad 1: Modelo estático. En este nivel se implementará la funcionalidad más simple, de modo que solo exista la posibilidad de realizar en un aula un único examen de la asignatura, con un solo modelo de examen, pero con la posibilidad de que efectúen el examen varios grupos distintos.”
2. “Grado de complejidad 2: Modelo estático. Este grado de complejidad abarca el primero, añadiendo la posibilidad de poder realizar varios modelos de examen de una misma asignatura. De modo que se podrá ubicar a los alumnos para que a su alrededor no tengan un compañero con el mismo modelo de examen.”
3. “Grado de complejidad 3: Modelo estático. El tercer nivel contempla la posibilidad

de que en un aula se puedan ubicar dos o más grupos de asignaturas diferentes, pero con un solo modelo de examen en cada asignatura.“

4. “Grado de complejidad 4: Modelo dinámico. En este nivel la implementación ya experimenta una dificultad mayor, pues al ser dinámico, el algoritmo para ubicar a los alumnos en el aula será más complejo. Así mismo, se podría tener un examen ubicado en un aula y con uno o varios grupos de la misma asignatura.“
5. “Grado de complejidad 5: Modelo dinámico. En este grado de complejidad, la aplicación permite ubicar a los alumnos de distintos grupos en una misma aula con más modelos de examen, pero de la misma asignatura.“
6. “Grado de complejidad 6: Modelo dinámico. Este último grado de complejidad se corresponde con la idea de tener en un mismo aula alumnos que realizan distintos exámenes de distintas asignaturas, con distintos grupos y distintos modelos de examen por cada asignatura.“



## Capítulo 5

# Histórico de la aplicación

Éste proyecto es una continuación del TFG de tres alumnos del año pasado. Ésto es así porque el proyecto no llegó a completarse. Mis compañeros del año pasado realizaron un gran trabajo en lo referente a los antecedentes, ya que buscaron información acerca de los primeros métodos de evaluación, empezados en la Edad Media, y su evolución. También llevaron a cabo una exhaustiva búsqueda para tratar de plasmar la importancia de éste proyecto poniendo como ejemplo a la Universidad Nacional de Educación a Distancia (UNED) con la “La Valija Virtual“, explicando detalladamente su función y el proceso que se desarrolla en la preparación y realización de exámenes. Otro punto que llevaron a cabo con gran nivel de detalle fue el análisis y la especificación, realizando una gran distinción entre los posibles modelos que se podían llegar a implementar, siendo éstos estático y dinámico, y todas sus posibles variantes dentro de esas dos posibilidades. Como ya anticipábamos anteriormente, realizaron un detallado estudio de los requisitos del proyecto, siendo éstos los correspondientes al sistema, usuarios, exámenes y el entorno. También cabe mencionar el arduo trabajo efectuado en la realización de todos los diagramas, tanto de uso, de Gantt, de procesos, del dominio, como de secuencia.

Pese al gran esfuerzo invertido por mis compañeros del año pasado, quedó pendiente mucho trabajo por hacer. Para realizar la implementación, basaron la aplicación en los lenguajes Javascript y HTML, ayudados de Express.js para la creación de plantillas. Y para el backend utilizaron MySQL como lenguaje para la BBDD, y JQuery junto con DHTMLX Scheduler para la realización de la funcionalidad del calendario. Después de descargar y analizar el repositorio, y crear una base de datos para interaccionar con la aplicación, se pudo apreciar que todo lo implementado eran partes separadas. Realizaron unas plantillas con ayuda de Express.js las cuales no estaban conectadas a nada, excepto el login, que eso sí funcionaba, pero el resto únicamente eran HTMLs estáticos, en el que se podían apreciar botones, subidas de archivos u opciones que no llevaban ni realizaban ninguna acción. Por otra parte, la implementación del DAO realizada, la cual no fue posible que se conectara a la BBDD, no es correcta. El patrón DAO trata de independizar lo máximo posible la lógica de la aplicación de la interacción con la base de datos, proponiendo como tal una interfaz con los métodos a implementar, a partir de la cual una serie de clases que heredarán de dicha interfaz tienen como objetivo implementar los métodos de la misma para la base de datos correspondiente. Pese a que los compañeros del año pasado conocían de la buena práctica del uso de interfaces DAO, no lo llevaron a cabo correctamente, puesto que no

realizaron ninguna interfaz DAO, sino que los DAOs que implementaron, son directamente las implementaciones de la base de datos, es decir, no se aplicó el patrón DAO en ningún momento, por lo que la implementación queda completamente acoplada a esta base de datos, suponiendo un problema de escalabilidad y de migración a otra base de datos.

Es por ello, entre otras cosas, que ha sido necesario continuar con el proyecto. Dado que la implementación, en gran parte no está conectada y en otros casos se ha realizado erróneamente, se ha decidido implementar la aplicación desde cero. Aún así, se han mantenido tecnologías del proyecto anterior como Node.js y Express. Aprovechando la oportunidad de empezar una aplicación, se realizaron búsquedas de arquitecturas para web, siendo Clean Architecture una de las más recomendadas para ello, y se decidió aplicarla al proyecto. Por otra parte, se ha iniciado una implementación que permita una alta escalabilidad, a la vez que ofrezca la posibilidad de reutilización de ese código en otros posibles proyectos siendo mínimos los cambios. Osea, se ha realizado la estructura de un proyecto válida para otros muchos, dando la posibilidad de comenzar un proyecto basándose en éste, extraer una parte del proyecto para ser utilizado en otro, o incluso extraer una parte del proyecto para ser sustituida por otra sin a penas tener impacto en el desarrollo ya realizado, gracias a la aplicación de Clean Architecture.

Para realizar todo ello, se ha dividido el proyecto en dos partes, frontend y backend. A la parte del backend se le ha aplicado, como ya se ha mencionado previamente, la arquitectura Clean Architecture, dividiendo el proyecto en cuatro capas, las correspondientes a la arquitectura.

El desarrollo ha consistido en primer lugar, como nexo entre el frontend y la lógica de negocio del backend, una API REST, siendo REST una interfaz que usa HTTP para la obtención de datos y el envío de ellos. Ésta API, realizada con Express.js, recibe las peticiones HTTP, y actúa en consecuencias llamando al controlador oportuno.

El controlador es el mediador entre la lógica de negocio y el enrutado de la aplicación, a la par que el encargado de la transformación de datos, ya sea realizar un *mappeo*, realizar un parseo, o crear un objeto. El controlador, tras la transformación de datos oportuna, realizará una llamada al caso de uso correspondiente.

Los casos de uso están divididos en carpetas por temática, y además, cada caso de uso ocupará un archivo completo, por lo que el nombre del archivo será el nombre del caso de uso correspondiente. Por su parte, los casos de uso son los encargados de la lógica de la aplicación, recayendo en éstos las validaciones de datos, siendo éstos los requisitos para el caso de uso, la obtención de datos, bien para comprobar, como puede ser la existencia de un usuario, con el fin de realizar la tarea del caso de uso seleccionado. Cada caso de uso realizará una serie de acciones en él, pudiendo ser entre ellas, la interacción con la base de datos.

Para la interacción con la base de datos se ha creado una interfaz con los métodos necesarios. De ahora en adelante, cuando a lo largo del TFG se hable de interfaz, hay que



tener en cuenta que en Javascript no existen interfaces de forma nativa. La clase creada para éste fin se trata de un mediador que te obliga a implementar esos métodos, pero se ha decidido llamarla interfaz por la función que tiene y para mayor comodidad en la lectura. Esta interfaz, es realmente importante, ya que es el nexo entre la lógica de la aplicación en la base de datos, realizando así el patrón DAO, y además, la inversión de dependencias para la interacción de los casos de uso con la base de datos cumpliendo el principio más importante de Clean Architecture, el principio de dependencias.

Con respecto a la implementación de la base de datos, se ha realizado en MySQL, utilizando para ello el ORM Sequelize. Este ORM es de gran importancia y utilidad, ya que desacopla la base de datos de la lógica de la aplicación, permitiendo tener varias bases de datos, siendo éstas de diferentes lenguajes, y además, aporta robustez y seguridad dado que evita las inyecciones de código SQL, entre otras.

Además de todo esto, se ha creado un modelo de gestión de errores. Este gestor de errores extiende de la clase Error, personalizándolo con un mensaje, definiendo el error, un código, el cual indica el nivel severidad, y por último, un alias, que aporta un extra de información para la solución del mismo, ya sea en el backend, como en el frontend.



# Capítulo 6

## Elección de las tecnologías

El desarrollo de una aplicación web requiere de una gran investigación. En primer lugar deberíamos analizar los requisitos de la aplicación, si será más de E/S o si requerirá de un gran cómputo. Una vez analizados los requisitos de la aplicación, deberemos analizar el o los lenguajes en los que deseamos desarrollar la aplicación, o si se tiene preferencia por algún framework el cual pueda implicar la selección del lenguaje indirectamente. También se deberían tener en consideración características como el rendimiento o si ese lenguaje es el más adecuado para el tipo de aplicación a desarrollar.

El eje central del desarrollo de la aplicación está basado en Javascript, adaptado tanto a la parte de backend como a la parte de frontend. Esto permite el uso de un único lenguaje para ambas partes, permitiendo crear una aplicación web desde el inicio hasta su finalización. Para poder utilizar Javascript en el lado del servidor, una buena elección, y además la más popular, es utilizar Node.js, un entorno de ejecución para el ámbito del backend el cual al ser no bloqueante y dirigido por eventos, resulta ser ligero y eficiente. Para facilitar el desarrollo del API Rest, se ha utilizado el framework Express, gracias al cual se minimiza el tiempo de creación y aporta robustez al proyecto. En todo proyecto es importante la validación de datos, para esa labor, se ha usado el middleware<sup>1</sup> express-validation, más concretamente Joi, el cual viene incluido. Joi ha resultado ser de gran utilidad para validar tanto la API como la lógica de negocio. Con respecto al almacenamiento de datos, se ha optado por utilizar MySQL, un sistema de gestión de base de datos relacional, o también llamados (RDBMS<sup>2</sup>), para las operaciones sobre la base de datos. También ha sido incluido el ORM 'Sequelize' con el fin de añadir una capa más entre la capa de negocio y la base de datos para de esta manera, tener la opción de cambiar la base de datos sin dificultad.

A continuación, se pasa a mencionar uno por uno los lenguajes y frameworks utilizados en el TFG.

---

<sup>1</sup>Un software que permite encadenar componentes seleccionados de una biblioteca o creados por el programador que implementan distintas tareas de procesamiento de peticiones HTTP que son comunes a muchas aplicaciones, tales como autenticación, logging, etc. Fuera del contexto de las aplicaciones web, este término tiene otros significados.

<sup>2</sup>Relational Database Management System: siglas en inglés de sistema de gestión de base de datos relacional

## 6.1. Javascript

En cualquier aplicación, es importante la elección del lenguaje o lenguajes en los que se va a desarrollar la misma. Para la parte del backend, se han barajado dos opciones, Java y Javascript. Pese a que Java es el lenguaje con el que aprendí a programar, y ofrece grandes funcionalidades y frameworks, la elección por Javascript se basó en que con ello era posible crear la aplicación completa con un único lenguaje, que es considerado el estándar para diseñar sitios web interactivos y que tiene una gran comunidad enfocada a web, la cual aporta numerosos frameworks y librerías que facilitan el desarrollo de la aplicación y aporta una gran cantidad de ejemplos y solución a posibles errores.

JavaScript es considerado el estándar por varias razones. Es un lenguaje que se integra perfectamente con HTML, CSS, etc. Resulta sencillo de aprender gracias a la gran documentación existente, la gran comunidad y la cantidad de ejemplos en la red. Facilita la incorporación de imágenes, botones, formularios y elementos multimedia entre otros. Tiene tipado dinámico y es multiparadigma, que quiere decir que permite usar distintos estilos de programación, en particular, funcional, dirigido por eventos y orientado a objetos basado en prototipos. Al ejecutarse en el lado del cliente, proporciona una respuesta más rápida que otros lenguajes. Además, tiene una amplia variedad de frameworks compatibles, como pueden ser NodeJS o React. Lo soportan los navegadores más conocidos como Google Chrome, Firefox, Safari, Internet Explorer u Ópera, además de los dispositivos móviles.

## 6.2. Node.js

Debido a la elección de Javascript como lenguaje para el backend, es necesario encontrar un entorno para la comunicación de Javascript con el sistema operativo. Actualmente la opción más fiable y consolidada es Node.js, aunque en un futuro próximo es muy probable que se deba tener en cuenta un nuevo entorno de ejecución llamado Deno, el cual sacó su primera versión en mayo y entre sus muchas ventajas destacan que, al igual que Node.js, utilizar el motor V8 de Chrome, que es un entorno seguro y dispone de una biblioteca más extensa.[6]

"Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello), basado en el lenguaje de programación ECMAScript, asíncrono, con entrada y salida de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google".[7]

Entre sus muchas ventajas para elegirlo como tecnología para el proyecto, caben destacar las siguientes ventajas y características:

- Debido a su arquitectura dirigida a eventos y con E/S asíncrona, favorece que sea

fácilmente escalable. Esto permite tener un gran número de conexiones de manera simultánea utilizando únicamente un servidor.

- Gracias a Node Package Manager (NPM) podemos añadir los módulos que nos sean necesarios de manera muy fácil.
- Al tratarse de un framework Open Source con una comunidad muy activa, favorece el acceso a muchos recursos como pueden ser manuales, librerías, foros, etc.
- Comodidad que ofrece al permitir desarrollar en el mismo lenguaje tanto en el backend como en el frontend.
- Al ser no-bloqueante y dirigido por eventos consigue que sea algo ligero y eficiente, por lo que aporta un muy buen rendimiento. El código Javascript se comunica con Node mediante una interfaz escrita en C++, esta interfaz se comunica con el motor V8 de Google y éste a su vez con el sistema. Es gracias a la utilización del motor V8 de google que el rendimiento se ve mejorado.

[19][33][25][26]

## 6.3. Express

El desarrollo de una aplicación web desde cero con Node.js no resulta trivial, entre otras cosas su creación, el enrutado, la gestión de cookies o sesiones... Afortunadamente hoy en día, se han desarrollado varios frameworks destinados a esa labor. Para este proyecto se han barajado tres posibilidades, Express, Hapi y Koa, aunque finalmente se ha seleccionado Express. [1]

Hapi es un framework consolidado basado en configuración sobre código, que es la principal diferencia con respecto a Express. Es un framework, al igual que Express, que es capaz de gestionar rutas, cookies, sesiones e incluso es posible añadir middleware. Entre sus muchas cualidades cabe destacar la limpieza en el código, la consistencia, la reutilización que permite y la escalabilidad. Pese a que el desarrollo de la aplicación hubiera sido posible con Hapi, se descartó debido a que es un framework más enfocado a una gran aplicación, que es donde realmente es posible sacarle partido, e impone ciertas restricciones en el desarrollo. En cambio, Express además de permitir escalar fácilmente la aplicación disponiendo de una gran flexibilidad para ello y proporciona una fácil integración de bibliotecas de terceros y productos intermedios.

Koa es un framework que permite crear una aplicación con un gran rendimiento. Este framework, creado por los desarrolladores de Express, posee como especiales características su ligereza, utiliza las funciones JS6 como generadores o `async/await`, tiene una mejor gestión de errores y aporta solidez debido a que resulta de fácil desarrollo escribir su middleware. Como se podrá observar, en conjunto, es un framework muy completo con unas grandes cualidades, pero relativamente nuevo, y es por eso por lo que aún no tiene una gran comunidad, y por tanto, existe más probabilidad de errores y menos

actualizaciones. Tampoco es compatible su middleware con el middleware de estilo express, por lo que si se deseara cambiar de framework por algún motivo, sería necesario una refactorización completa. Además, sus generadores no son compatibles con ningún otro tipo de middleware de framework de Node. Es por estos tres motivos por los que no se ha seleccionado este framework para la aplicación.

Express es un framework destinado a facilitar la creación de aplicaciones web en menor tiempo, debido a las funcionalidades que ofrece. Entre ellas, está el enrutamiento, creación y utilización de middleware, la gestión de cookies, motores de plantilla<sup>3</sup>, sesiones...

A continuación se muestran algunas de las principales ventajas que se obtiene al utilizar Express.

- La primera de las ventajas a tener en cuenta sobre Express es que desarrollar una aplicación con este framework resulta fácil y rápido.
- Gracias a que Express está basado en Connect, en el framework ya tiene incorporado todas las funcionalidades del módulo http.
- Otra de las ventajas que ofrece Express es que resulta sencillo de personalizar y de configurar.
- Un ejemplo de ello es que permite personalizar un end-point añadiendo el middleware que deseemos.
- Como ya hemos anticipado en la definición y el segundo punto, permite definir rutas basadas en URL y métodos HTTP.
- Para ello, cuenta con una clase que facilita el enrutado, llamada 'Router'.
- 'Router' es un componente de middleware y direccionamiento completo ya que da la posibilidad de crear manejadores de rutas montables y modulares.
- Una de las funcionalidades extra que ofrece Express es que gracias al middleware del que dispone puede realizar otras tareas además de las peticiones de solicitud y respuesta, como puede ser el manejo de errores.
- Otro de los puntos fuertes de Express es que permite crear un servidor API REST.
- Esto permite el desacople y una alta reutilización de la lógica de negocio para los diferentes frontales, como aplicaciones web, nativas o software de escritorio.
- Para una aplicación web con posibilidades de crecer, resulta importante la escalabilidad.
- Express permite que la aplicación sea fácilmente escalable gracias a que tiene varios clústers con Express funcionando.

---

<sup>3</sup>un sistema de procesamiento que genera documentos a partir de unos datos y unas plantillas. En el contexto de una aplicación web, los documentos de salida son contenido HTML o XML, los datos de entrada suelen venir de una base de datos y las plantillas suelen contener una mezcla de HTML / XML e instrucciones de programación.

- Además, permite conectar con la base de datos, ya sea MongoDB o MySQL, de una manera muy fácil.
- Se trata de un framework OpenSource, con una comunidad muy activa. Esto ha repercutido en convertir a Express en un framework consolidado, siendo además la opción más popular.
- Por último, Express es fácilmente integrable con otros motores de plantilla como pueden ser Vash, EJS o Jade.

[3][17][18][28][42]

## 6.4. Express-validation

En cualquier aplicación, es necesario validar datos con el fin de proporcionarla más robustez. Para ello, hemos tenido en cuenta a la hora de utilizar cualquier librería que fuera compatible con Express. Por ese motivo, se han barajado dos opciones, `express-validation` y `express-validator`. Ambas opciones validan datos en los endpoint<sup>4</sup> creados en Express, y ambos tienen multitud de opciones de validación como email, expresiones regulares, tamaño de texto o gestiones de errores, entre otras. Para el caso de `express-validator`, se valida en el mismo endpoint todo lo que deseemos. Finalmente, la opción escogida ha sido `express-validation`, ya que además de validar datos en los endpoint, permite crear funciones. Éstas funciones serán pasadas como parámetro a los endpoint, lo cual nos permite una gran reutilización del código para casos en los que la validación de datos sea la misma, un claro ejemplo de esto en la aplicación es la validación para los distintos tipos de usuarios existentes. Todo esto es posible gracias a que ha sido incorporada la librería Joi a `express-validation`, la cual nos permite no solo validar a nivel de API, sino en el resto de la aplicación.

`Express-validation` se trata de un middleware encargado de validar el modelo de datos de entrada a la API. Para realizarlo, valida una solicitud de entrada y devuelve una respuesta con errores en el caso de que los datos introducidos no sean correctos.

Permite validar parámetros como pueden ser ‘params’, ‘query’, ‘cookies’, ‘headers’ o ‘body’.

Su utilización es sencilla, ya que utiliza Joi para ello. Joi es una librería que permite crear funciones validadoras para verificar modelos de datos. Estos datos vienen dados por el formato JSON, que es un estándar de comunicación entre sistemas. Crea una función validadora de los datos que será pasada como parámetro al endpoint para que el endpoint los valide.

---

<sup>4</sup>un URL en el que se puede acceder a un servicio web; un mismo servicio puede tener múltiples endpoints. Por extensión, a veces se usa este término para referirse al código que procesa las peticiones enviados a un endpoint.

[16]

## 6.5. MySQL

Para el caso de la base de datos, se analizó la posibilidad de realizar un modelo relacional con MySQL o uno no relacional con MongoDB. Dadas las diferencias entre lo relacional y no, se descartó la idea de una base de datos no relacional debido a que no se va a tener una cantidad de datos muy grande y es muy improbable que haya habitualmente cambio en los esquemas de las tablas. Pese a ello, la aplicación está diseñada para ser lo más independiente posible del tipo de base de datos, pudiendo ser esta no relacional si fuese el caso. MySQL[8], como ya se ha anticipado, es un sistema de gestión de base de datos relacional basado en el modelo cliente-servidor. Los datos están distribuidos en diferentes tablas compuestas por filas y columnas. Estas columnas y filas están conectadas mediante relaciones. Es de código abierto y utiliza SQL como lenguaje para la comunicación con la base de datos. Al ser una base de datos multiplataforma, puede ejecutarse en diferentes sistemas operativos.

La primera característica destacable de MySQL es la actuación, que proporciona un gran rendimiento en situaciones delicadas sin que los datos resulten expuestos. Además, cumple con las características ACID (atomicidad, consistencia, aislamiento y durabilidad), que garantizan el procesamiento de transacciones y la precisión de datos en situaciones desfavorables. También proporciona un fuerte mecanismo de seguridad como pueden ser el cifrado de contraseñas o la autorización de cuentas de usuario. Al utilizar SQL como lenguaje para la base de datos, MySQL permite el uso de componentes SQL avanzados como vistas, procedimientos y funciones.

## 6.6. Sequelize

Según la documentación, “Sequelize es un ORM de Node.js basado en promesas<sup>5</sup>[14] para Postgres, MySQL, MariaDB, SQLite y Microsoft SQL Server. Cuenta con un sólido soporte de transacciones, relaciones, carga ansiosa y perezosa, replicación de lectura y más”[10].

Sequelize dispone de ciertas características destacables como el hecho de que funciona con promesas, tiene soporte de transacciones y permite validar los modelos. Además, dispone de soporte para MySQL, SQLite, PostgreSQL y Microsoft SQL Server.

La utilización de Sequelize ofrece múltiples beneficios. Uno de ellos es la prevención de inyección de código, gracias a que las consultas se realizan mediante funciones y no con consultas SQL directamente. Otra característica es que permite escribir código orientado

---

<sup>5</sup>. Es un objeto que representa la terminación o el fracaso de una operación asíncrona"[5] la cual no se sabe cuando se va a resolver. Se utilizan para llevar a cabo código asíncrono así como para tareas pesadas, ya que gracias a este enfoque, permite no bloquear el código y por ende, la interfaz



a objetos para los modelos y abstrae el proceso CRUD permitiendo escribir código más declarativo. La reducción de tiempo es otra de las características de las que dispone, ya que minimiza el tiempo de desarrollo puesto que no es necesario escribir las consultas SQL ya que dispone de unas llamadas a funciones Javascript que realizan esa tarea. Además, cuenta con una interfaz segura para llevar a cabo la interacción con la base de datos.

## 6.7. Webstorm

Una vez analizados los conocimientos, la experiencia y la habilidad en varios entornos utilizados tanto a lo largo de los cursos en la universidad como en la vida laboral, se han barajado Eclipse, Visual Studio Code y Webstorm como IDEs para el proyecto. De estos tres IDEs se ha tenido en cuenta la experiencia del desarrollador con el IDE, la interacción e integración de las librerías seleccionadas, documentación, integración de frameworks y soluciones de código abierto.

En base a estos requisitos, se ha optado por desarrollar la aplicación en el entorno de desarrollo WebStorm, que además de contener las cualidades mencionadas, proporciona un buen análisis estático del código, análisis de código activo, ofrece un mejor soporte para entornos de JavaScript ayudando y dando facilidades al desarrollador, y dispone de una interfaz amigable.

Este IDE pertenece a la familia JetBrains, la cual dispone de una amplia gama de IDEs especializados para casi todos los lenguajes, el más conocido IntelliJ IDEA para Java.



# Capítulo 7

## Arquitectura

### 7.1. Clean Architecture

Clean Architecture[30][40][29][13][32][38] es un patrón arquitectural de diseño ágil de software propuesto hace una década por Robert C. Martin (también llamado "uncle Bob"), el autor de la conocida colección de principios de diseño software denominada SOLID. Consiste en una arquitectura que se basa en estructurar el código en capas contiguas. Estas capas solo podrán comunicarse con las capas que tengan a ambos lados.

Esta arquitectura tiene como norma general la regla de dependencias. Esta regla sostiene que las dependencias del código fuente solo pueden apuntar hacia adentro, es decir, nada de lo que pueda contener una capa externa será visible para una capa interna. Con el fin de no afectar a las capas interiores, no se deberían usar estructuras de datos definidas en las capas exteriores en ellas. Lo único que se permitirá pasar de una capa a otra serán datos con una estructura simple.

Las capas básicas de las que se compone Clean Architecture son Framework & Drivers, Interface Adapters, Application Business Rules y Enterprise Business Rules, ordenadas de la capa situada más al exterior a la capa situada más al interior.

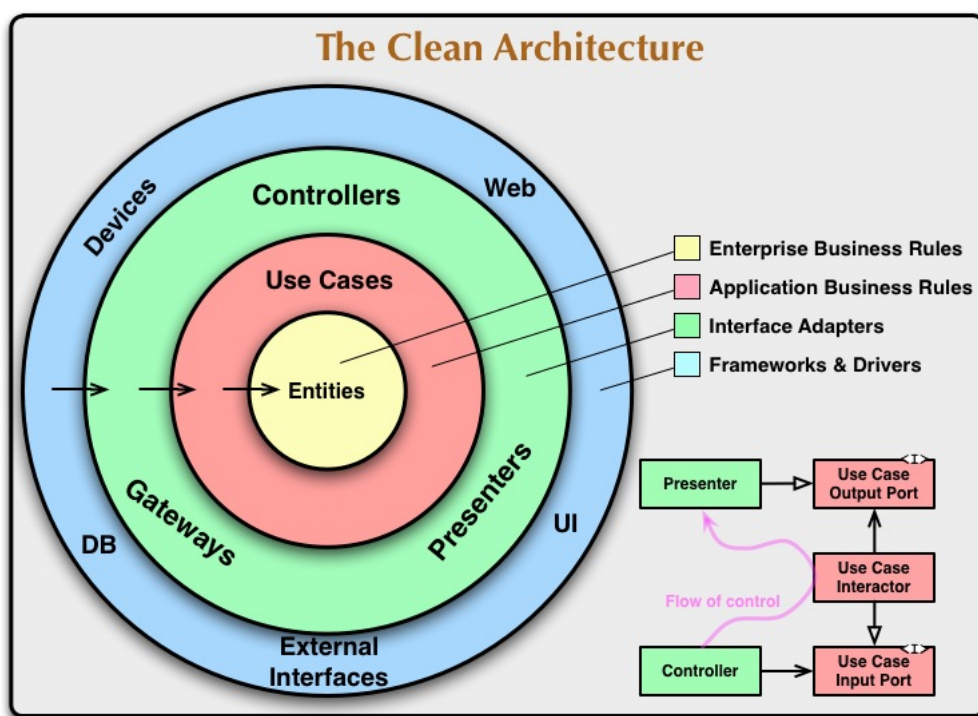


Figura 7.1: Imagen Arquitectura Limpia

La capa Frameworks & drivers está compuesta de las herramientas y frameworks, tales como la base de datos o los frameworks web que van a ser utilizados en la aplicación.

La capa Interface Adapters es la que se va a encargar de transformar los datos en algo que entienda la siguiente capa que los vaya a utilizar, además contendrá los controladores, presentadores y accesos a servicios de terceros.

La capa Application Business Rules es la que contiene los casos de uso y, por tanto, la lógica de la aplicación. En ella se definirán los datos de entrada, los datos de salida y el comportamiento que tendrá el sistema.

La última capa es Enterprise Business Rules. En esta capa se encuentran las entidades, que contienen las reglas de negocio. Estas entidades deberán componerse de las funciones más básicas. Además, una entidad puede ser usada por uno o más componentes, por lo que serán independientes y no deberán cambiar por influencia de elementos externos.

Siempre habrá casos en los que alguna capa interna necesitará algún dato de una capa externa, pero, como ya hemos visto previamente, una capa interna no puede ver a una capa externa por la regla de dependencias. Para ello se utiliza el principio de inversión de dependencias, uno de los principios SOLID. Este principio se basa en dos reglas, la primera es que la capa exterior no debe depender de la capa interior y que ambas capas deben depender de las abstracciones. La segunda regla es que las abstracciones no deben

depender de los detalles, sino los detalles de las abstracciones.

Para ilustrar esto se va a usar el siguiente ejemplo: es posible que un caso de uso (capa Application Business Rules) necesite uno o varios datos de una base de datos o de un servicio (capa Frameworks & drivers) y, como ya hemos anticipado, por la regla de dependencias, no es posible solicitar datos directamente a la base de datos o al servicio. Para no romper esta regla, se creará una interfaz en la capa Application Business Rules que definirá los datos necesarios de entrada y de salida. Esa interfaz será implementada por la capa Frameworks & drivers, ciñéndose a lo definido en la interfaz. De esta manera, cada capa sigue sin saber nada de la otra y no se rompe la regla de dependencias manteniendo así el flujo unidireccional hacia adentro.

## 7.2. Por qué usar Clean Architecture

Tras realizar una investigación sobre arquitecturas enfocadas al entorno web, se ha llegado a la conclusión de que Clean Architecture es una de las mejores opciones actualmente.

Clean Architecture es una buena opción por múltiples razones. La primera de ellas es la independencia, puesto que cada capa tiene su propio paradigma como si de una aplicación se tratase, y el resto de capas no se ven afectadas por su contenido. Al ser las capas independientes, resulta muy sencillo modificar cualquier parte del código, como puede ser la interfaz de usuario o la base de datos.

Otra de las razones es que es fácilmente testeable. Gracias a la independencia entre capas, se favorece que sea más rápido la revisión de una funcionalidad concreta. Además, por esa misma independencia, es posible crear test de end-to-end en cada nivel. De esta manera, se puede testear cada capa por separado y se pueden crear objetos que simulen la entrada/salida de datos de otra capa para poder testear una nueva funcionalidad sin necesidad de usar datos reales.

El hecho de que la aplicación esté estructurada en capas favorece la organización del código y, por tanto, se facilita la búsqueda de funcionalidades y su navegación a través del scaffolding<sup>1</sup>.

La más importante de las razones para usar Clean Architecture es el desacoplamiento. Al ser las capas independientes las unas de las otras evitamos que, ante futuros cambios como puedan ser un cambio de la base de datos o el de algún framework, estos afecten a nuestra lógica de negocio. Además, Clean Architecture favorece el poder desarrollar el código en diferentes tecnologías y su reutilización en otras aplicaciones parecidas.

---

<sup>1</sup>Estructura de carpetas y archivos de una aplicación



# Capítulo 8

## Instalación

### 8.1. Introducción

Para llevar a cabo la implementación de la aplicación, se ha dividido en dos partes, el frontend y el backend.

Para proceder a su desarrollo, se decidió utilizar Javascript como único lenguaje en la aplicación, debido a ello, una buena elección es Node.js como entorno de ejecución para la capa del servidor. Con el fin de facilitar la implementación de la API, se ha optado por utilizar Express, un framework que facilita la creación de aplicaciones web, así como enrutado y utilización de middleware entre otros. Para la validación de datos en la API, se ha utilizado la librería Joi, la cual viene incluida en express-validation con la que es posible crear reglas de validación.

A continuación se va a explicar cómo se realizaron las instalación y las referencias a las páginas que resultaron de guía o de página de descarga.

### 8.2. Webstorm

El primer paso que se llevó a cabo fue la instalación del entorno especializado en Javascript, Webstorm.

Para la instalación de Webstorm, recurrimos a la página JetBrains, la cual facilita el poder crearse una cuenta totalmente gratuita gracias al correo de estudiante de la UCM, y poder así, disfrutar de cualquier IDE de la familia JetBrains.

Es posible mediante la página directa de Webstorm, descargarse el IDE e instalarlo. Pero, aprovechando la cuenta que el correo UCM nos ofrece, se ha descargado una aplicación llamada JetBrains ToolBox. Esta aplicación aún más de 15 IDEs especializados de JetBrains, dado lo cual, es posible instalarse cualquier IDE que contenga de JetBrains únicamente haciendo click en **Install**. Desde esta misma aplicación, es posible actualizar

un IDE previamente instalado, y abrir un proyecto para el cual estemos utilizando alguno de los IDEs.

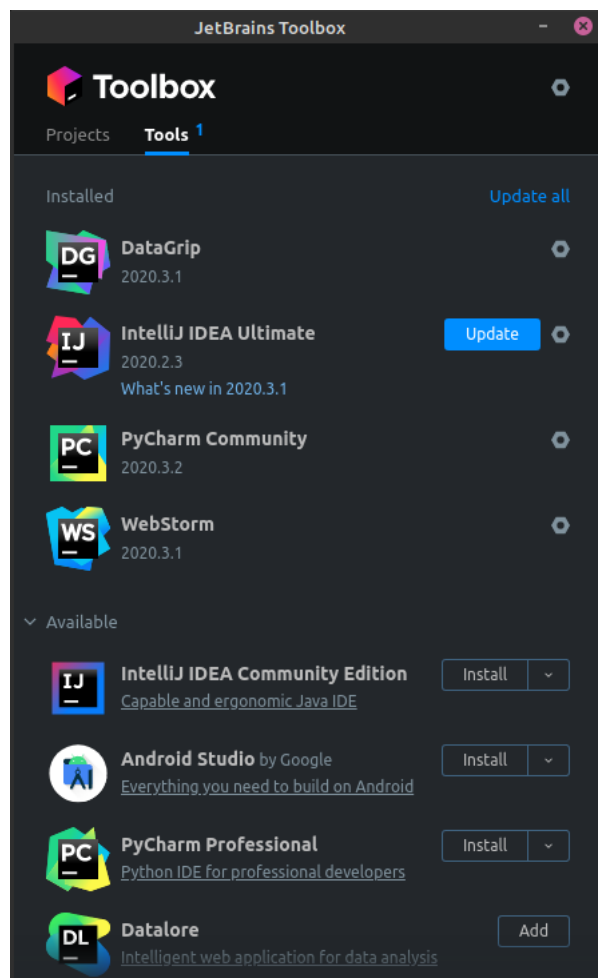


Figura 8.1: Imagen ToolBox

Una vez instalado Webstorm, se configuró el archivo de ejecución. Para este proyecto se configuró el archivo `server.js` como el archivo de ejecución del proyecto, se especificó la carpeta del proyecto y la versión de Node.js.



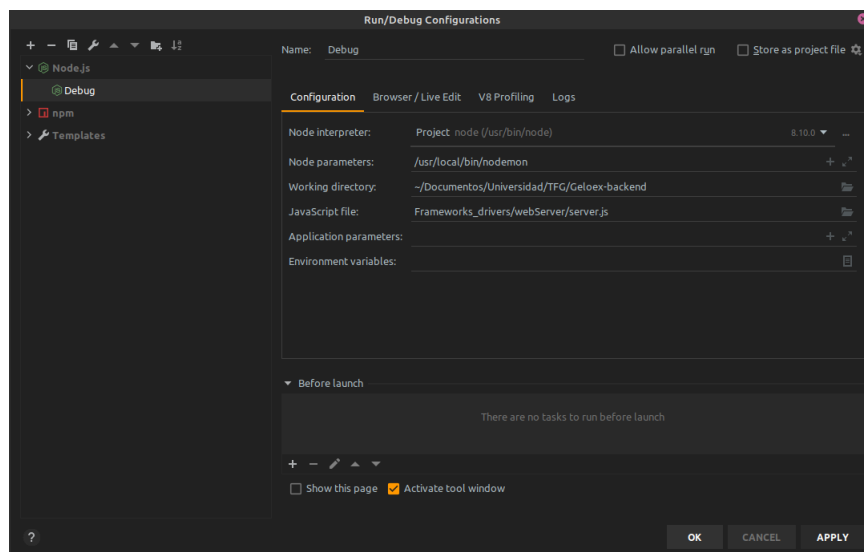


Figura 8.2: Imagen configuración

Otro aspecto de la configuración en Webstorm fue el de la sincronización con el repositorio de GitHub, el cual se realizó mediante ssh. Gracias a esta sincronización, aparece en Webstorm una ventana extra en la parte de abajo en la cual pueden verse, entre otras cosas, los archivos que no han sido commiteados, realizar commits, rollbacks, refresh, push, pull o mostrar diferencias entre un archivo antes y después de haberlo modificado.

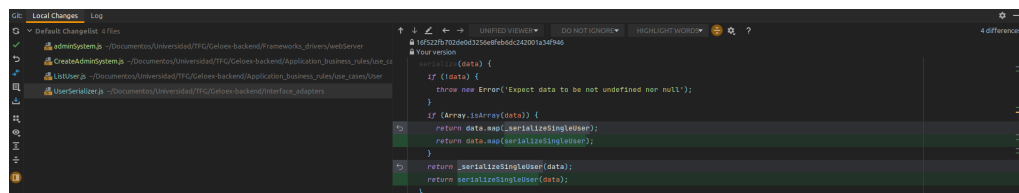


Figura 8.3: Imagen Git

## 8.3. Instalación Node.js

Como ya se anticipó en el capítulo sobre tecnologías, era necesario la instalación de un intermediario entre el sistema operativo y Javascript, se ha seleccionado la solución más popular, Node.js.

Puesto que el sistema operativo utilizado para la realización del proyecto es una distribución de Linux, se optó por instalar Node.js mediante el comando `sudo apt install nodejs`, del cual no se obtuvo ningún problema.

## 8.4. Instalación npm

npm es un gestor de paquetes para Node.js cuando el lenguaje de desarrollo es Javascript.

Dado que Node.js está formado por módulos, npm facilita el agregar dependencias, gestionar tareas y distribuir y gestionar los paquetes que irán siendo necesarios a lo largo del proyecto. Cuando npm es instalado en el proyecto, se creará una carpeta llamada `node_modules` en la cual se almacenarán las dependencias de la aplicación, muchas estarán incluidas por Node.js, y otras serán las que se vayan agregando durante el proyecto. Otro archivo relacionado con npm es `'package.json'`, este archivo es posible crearlo a mano o mediante el comando `npm init`. Es archivo contendrá las dependencias que hemos instalado, así como el nombre, la versión, o scripts, como puede ser el de la ejecución de la aplicación.

Para la instalación de npm, se ha ejecutado el comando `sudo apt-get install npm`. A continuación se muestra el `package.json` de la aplicación, sin ser ésta la versión final.

```
1  {
2    "name": "geloex-backend",
3    "version": "0.0.0",
4    "private": true,
5    "scripts": {
6      "start": "node ./Frameworks_drivers/webServer/server.js",
7      "start-es6": "node --experimental-modules ./bin/www.mjs",
8      "lint": "eslint . -c ./eslinttrc.json",
9      "lint-node": "eslint . --ext .js --env node --ignore-pattern node_modules/ -c ./eslinttrc.json"
10   },
11   "husky": {
12     "hooks": {
13       "pre-commit": "npm run lint",
14       "pre-push": "npm run lint",
15       "post-merge": "npm install",
16       "post-rewrite": "npm install"
17     }
18   },
19   "dependencies": {
20     "cookie-parser": "~1.4.4",
21     "debug": "~2.6.9",
22     "express": "~4.16.1",
23     "express-validation": "^3.0.2",
24     "husky": "^3.1.0",
25     "morgan": "~1.9.1",
26     "mysql2": "^2.1.0",
27     "sequelize": "^6.3.4"
28   },
29   "devDependencies": {
30     "eslint": "^6.8.0",
31     "eslint-config-airbnb-base": "^14.1.0",
32     "eslint-plugin-import": "^2.20.2"
33   }
34 }
```

Figura 8.4: Imagen Package.json

## 8.5. Instalación y creación del proyecto mediante Express

En la sección Elección de Tecnologías ya se describió que Express es un framework destinado a facilitar la creación de aplicaciones. Y que entre sus muchas cualidades cabe destacar el enrutamiento, creación y utilización de middleware, la gestión de cookies, motores de plantilla, gestión de sesiones.

Para la instalación de Express, se ha recurrido de nuevo a la línea de comandos, en este caso hemos utilizado `npm install express -save` ya que queríamos incluirlo en la lista de dependencias.

A continuación hemos creado el proyecto, para crear un proyecto con Express se ha utilizado el comando `express Geloex`. Este comando genera una serie de carpetas y ficheros por defecto. Entre ellos cabe mencionar `package.json`, `app.js` (archivo de ejecución de la aplicación) o la carpeta `routes`, en la cual crearán los archivos para las rutas).

## 8.6. Refactorización del scaffolding

Como se vió en la sección anterior, cuando creamos una aplicación utilizando Express, se crean una serie de archivos y carpetas con archivos por defecto.

La refactorización ha surgido para facilitar el desarrollo de la aplicación haciendo uso de la arquitectura Clean Architecture. En la sección Arquitectura, se vió como Clean Architecture se dividía en cuatro capas. La capa Enterprise business rules, encargada de albergar las entidades y las reglas de negocio básicas. La capa contigua es Application business rules, cuya labor es contener los casos de uso, y por tanto, las reglas de negocio. La siguiente es Interface adapters, esta capa es la encargada de la transformación de datos tanto para la capa Application business rules, como para la capa más externa, Frameworks & drivers. Y por último, Frameworks & drivers, que contiene las herramientas y frameworks utilizados para la aplicación.

Esta refactorización ha consistido en la creación de una carpeta por capa. Pese a la refactorización, fue una ardua tarea de meses llegar a llevar a cabo la primera funcionalidad de la aplicación correctamente y respetando la arquitectura, debido a que los conocimientos en arquitectura no eran lo suficientemente buenos como para que fuese fluido, y continuamente se violaba el principio de la arquitectura, la regla de las dependencias hacia abajo.

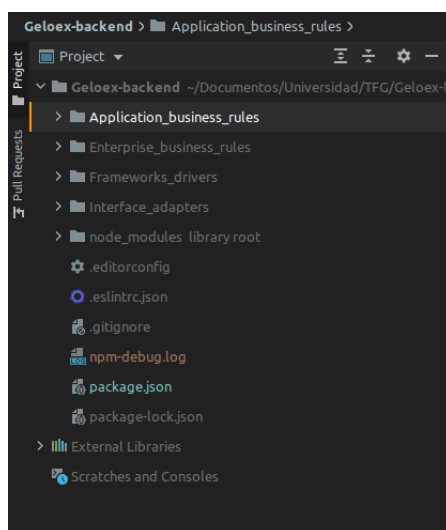


Figura 8.5: Imagen Scaffolding del proyecto

Como se aprecia en la imagen, el scaffolding está compuesto, como acaba de mencionarse previamente, por las cuatro carpetas representantes de Clean Architecture, `Application_business_rules`, `Enterprise_business_rules`, `Frameworks_drivers` e `Interface_adapters`. Además de ellas, contiene una carpeta llamada `node_modules`, en la cual aparecerán todas las dependencias que han sido instaladas por defecto en la instalación de Node.js, y también las instaladas durante el desarrollo.

Además de carpetas, está compuesto también por varios archivos. El archivo `gitignore` está relacionado con git, este archivo contiene todo lo que no se quiere subir a git, de esta manera, todo lo que esté incluido en este archivo, será ignorado en el momento de hacer un commit para subir contenido a la plataforma.

```
1  /routes/
2  .class
3
4  # IntelliJ IDEA
5  .idea
6  *.iml
7
8
9  # Logs
10 logs
11 *.log
12 npm-debug.log*
13
14 # Runtime data
15 pids
16 *.pid
17 *.seed
18 *.pid.lock
19
20 # Dependency directories
21 node_modules/
22 jspm_packages/
23
24 # Output of 'npm pack'
25 *.tgz
26
27 notes.md
```

Figura 8.6: Imagen gitignore

Otro archivo es el `package.json`, del que ya se ha hablado. Existen dos apartados, `dependencies` y `devDependencies`. En el apartado `dependencies`, han sido agregadas las librerías y frameworks en las que se basa la aplicación y que van acompañados de su versión. En el caso de `devDependencies`, es el apartado en el que se agregan paquetes junto con su versión, pero que no son necesarios para el funcionamiento de la aplicación. El `package.json` también es usado para lanzar scripts, como el de la ejecución de la aplicación, y que deben ser añadidos en el apartado `scripts`. También contiene el nombre de la aplicación o el de la versión.

```
1 {
2   "name": "geloex-backend",
3   "version": "0.0.0",
4   "private": true,
5   "scripts": {
6     "start": "node ./Frameworks_drivers/webServer/server.js",
7     "start-es6": "node --experimental-modules ./bin/www.mjs",
8     "lint": "eslint . -c ./.eslintrc.json",
9     "lint-node": "eslint . --ext .js --env node --ignore-pattern node_modules/ -c ./.eslintrc.json"
10  },
11  "husky": {
12    "hooks": {
13      "pre-commit": "npm run lint",
14      "pre-push": "npm run lint",
15      "post-merge": "npm install",
16      "post-rewrite": "npm install"
17    }
18  },
19  "dependencies": {
20    "cookie-parser": "~1.4.4",
21    "debug": "~2.6.9",
22    "express": "~4.16.1",
23    "express-validation": "^3.0.2",
24    "husky": "^3.1.0",
25    "morgan": "~1.9.1",
26    "mysql2": "^2.1.0",
27    "sequelize": "^6.3.4"
28  },
29  "devDependencies": {
30    "eslint": "^6.8.0",
31    "eslint-config-airbnb-base": "^14.1.0",
32    "eslint-plugin-import": "^2.20.2"
33  }
34 }
```

Figura 8.7: Imagen Package.json

## 8.7. Instalación y conexión MySQL

La base de datos utiliza el lenguaje MySQL. En principio, parecía que la instalación iba correctamente, pero al intentar conectarse a ella, aparecieron problemas con la conexión debido al usuario, tras varios cambios de contraseña, e incluso quitarla, seguía sin ser posible la conexión. Se instaló MySQL dos veces, por si el problema era de no haber seguido bien la guía o si se hubiera saltado algún paso, pero no fue así. Fue a la tercera, con otra guía[8] cuando al instalar, se conectó sin problemas.

# Capítulo 9

## Implementación

### 9.1. Introducción

Como en toda aplicación, es necesario la realización de un profundo análisis y especificación. Este proyecto no lo tiene debido a que los alumnos del TFG[34] de hace dos años hicieron un gran trabajo de análisis y especificación y se ha basado en ese trabajo para llevar a cabo la implementación de éste.

Con el fin de facilitar la comprensión del proyecto, el sistema de carpetas contenidas por el proyecto se ha organizado en función a las capas que contiene Clean Architecture. En base a esto, el proyecto dispondrá de cuatro carpetas, `Enterprises_business_rules`, `Application_bussiness_rules`, `Interface_adapters` y `Framework_drivers`, procurando implementar en cada capa lo correspondiente.

### 9.2. Implementación de la especificación

La implementación de la gestión de exámenes que lleva a cabo este proyecto, se ha basado en los cuatro roles para usuarios propuestos por el TFG[34] de hace dos años, Administrador de Exámenes, Administrador de Sistemas, Profesor y Alumno.

Tras el análisis de los roles, crearon un conjunto de casos de uso para cada rol, siendo éstos las acciones que podría realizar cada usuario. A continuación, se va a comentar cada caso de uso y si se ha implementado.

Especificaron que un Administrador de sistema, encargado de gestionar los usuarios, tendría como casos de uso el poder crear un usuario, eliminar un usuario, y buscar un usuario. En lo referente al administrador de sistemas, los tres casos de uso han sido implementados, y además de ello, también se le ha otorgado la posibilidad de modificar cualquier campo de un usuario ya creado, añadir un nuevo rol a ese usuario, modificar un rol y eliminar un rol siempre y cuando quede mínimo un rol asignado al usuario.

El administrador de exámenes por su parte, encargado de la gestión de exámenes, le asignaron la tarea de indicar la fecha, la hora y el aula a un examen. En la implementación en este proyecto además de asignar fecha y hora, también es el encargado de crear un examen, eliminar un examen, listar los exámenes, buscar un examen o modificar cualquier

campo de un examen, pudiendo ser éstos la duración, el grupo, la asignatura, el estado, el profesor o el aula. También será el encargado de añadir una nueva asignatura a la plataforma, para el caso en el que se haya creado una nueva asignatura y sea necesario crear el examen. Y al igual que puede crear asignaturas, también podrá modificarlas o eliminarlas. Además, generará el calendario de exámenes.

Por su parte, al profesor le encomendaron las tareas de que sea posible consultar un examen, eliminar un modelo de examen, subir un modelo de examen, recuperar las soluciones a un examen, agregar tiempo de duración a un examen, que pueda dar comienzo a un examen, que sea capaz de vigilar un examen y de terminar un examen. De todo ello, se ha llegado a implementar la posibilidad de agregar tiempo de duración a un examen, la consulta de un examen del cual sea profesor, obteniendo como datos la asignatura, día de la fecha del examen, tiempo de duración, el o las aulas, el grupo y el estado. El comienzo, vigilancia y terminación de un examen se ha llevado a cabo mediante la modificación del estado de un examen. Además, también es posible mostrar todos los exámenes de los que es el profesor titular.

Para el caso del alumno, propusieron como casos de uso únicamente la identificación del usuario y la entrega electrónica. Para el caso del alumno, se ha implementado el poder listar los exámenes de las asignaturas en las que está matriculado.

Aunque no se ha mencionado previamente, la identificación completa de un usuario, entiendo como tal la lectura de la tarjeta del carnet universitario, no se ha llegado a implementar. Lo que la aplicación si es capaz de distinguir es el rol de un usuario, y las tareas que son capaces de realizar. Un ejemplo de ello es la de creación de un usuario, la parte encargada de la creación de un usuario recibirá los datos del usuario a crear, a la par que un identificador de usuario, el cual permitirá saber el rol que posee el usuario que ha solicitado crear el usuario, y a no ser que tenga el de administrador de sistemas, no podrá crear el usuario.

Con respecto a la reserva de las aulas para exámenes, se realiza con la acción del administrador de exámenes de generar calendario. Este calendario se genera en función del número de aulas y la distribución de las mismas, del número de alumnos que se estime que se va a presentar al examen, el número de exámenes y el número de modelos que dispone un examen.

En lo que respecta a la asignación de asientos, se realiza en el momento del examen. Aunque no exista un logueo como tal, es posible simularlo llamando a la función correspondiente encargada de la distribución de asientos que, pasándole el identificador del alumno y el identificador del examen, calculará el asiento que debe ocupar.

### 9.3. Enterprise \_\_business\_\_ rules

Ya se ha mencionado en otra sección, que esta carpeta hace referencia a la capa Enterprise Business Rules de Clean Architecture, y por ello su nombre.



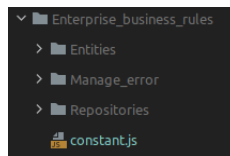


Figura 9.1: Imagen Enterprise\_business\_rules

Esta capa es la encargada de alojar las entidades, entendiendo como entidad toda regla de negocio que resulta ser crítica para el aplicación. En esta carpeta se encuentran otras tres carpetas, Entities, Repositories y Manage\_error. Conste que todas las carpetas contienen entidades, únicamente se han dividido para mayor comodidad y agilidad en la navegación.

También existe un archivo llamado constant.js. Este archivo almacena las constantes que van a ser utilizadas a lo largo del desarrollo. Actualmente contiene los roles de usuario, los distintos grupos existentes por curso, y los estados de un examen, pero es probable que cuando vaya avanzando el desarrollo, sea necesario incluir otro tipo de constantes.

```
constant.js
1  const ROLES = {
2    TEACHER: 'TEACHER',
3    STUDENT: 'STUDENT',
4    ADMIN_SYS: 'SYSTEMS ADMINISTRATOR',
5    ADMIN_EXAM: 'EXAMS ADMINISTRATOR',
6  };
7
8  const GROUPS = {
9    A: 'A',
10   B: 'B',
11   C: 'C',
12   D: 'D',
13   E: 'E',
14   F: 'F',
15   G: 'G',
16   H: 'H',
17   I: 'I',
18   V: 'V',
19   DG: 'DG',
20 };
21
22 const STATE = {
23   InProgress: 'In progress',
24   Finalized: 'Finalized',
25   ToDo: 'To do',
26 };
27
28 module.exports = {
29   ROLES, GROUPS, STATE,
30 };
```

Figura 9.2: Imagen constantes

A continuación, se va a pasar a explicar el contenido de las tres carpetas.

### 9.3.1. Entities

En esta carpeta se van a encontrar todas las entidades que representan una definición para la aplicación, como puede ser un profesor, un examen o un espacio.

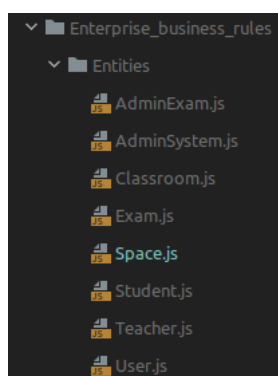
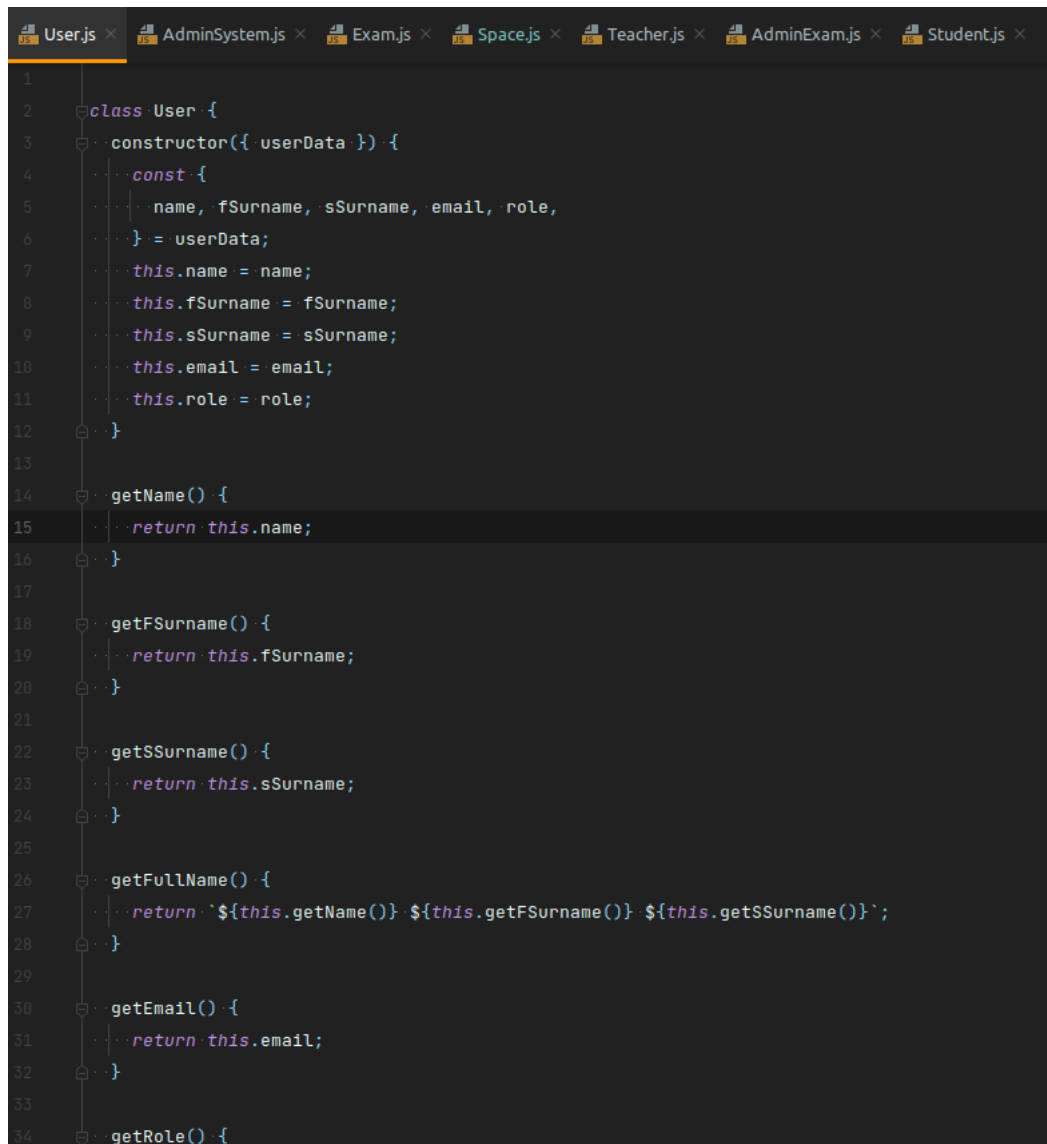


Figura 9.3: Imagen Entidades

Se va a comenzar con la entidad **User**. Esta entidad se trata de una clase abstracta de la que derivan los distintos tipos de usuarios de los que está compuesta la aplicación. Esta clase es la encargada de crear un usuario exigiendo al menos el nombre, apellidos, el email y el rol del usuario, pudiendo ser éste cada uno de los distintos tipos de usuarios que existen en la aplicación. Se ha tenido en consideración que un usuario pueda tener más de un rol asignado, pudiendo de esta forma dar la posibilidad que un usuario sea profesor, a la vez que administrador de exámenes. Otras de las funcionalidades que ofrece esta clase es el poder consultar y modificar cada dato del que está compuesto un usuario mediante los habituales getters y setters para cada dato. Y además de la consulta y modificación de datos, es posible la conversión de datos de un usuario a un formato JSON. El resto de entidades que heredan de la clase **User** son **AdminExam**, **AdminSystem**, **Teacher** y **Student**.

The image shows a code editor with several tabs at the top: 'User.js', 'AdminSystem.js', 'Exam.js', 'Space.js', 'Teacher.js', 'AdminExam.js', and 'Student.js'. The 'User.js' tab is active, displaying the following JavaScript code:

```
1
2 class User {
3   constructor({ userData }) {
4     const {
5       name, fSurname, sSurname, email, role,
6     } = userData;
7     this.name = name;
8     this.fSurname = fSurname;
9     this.sSurname = sSurname;
10    this.email = email;
11    this.role = role;
12  }
13
14  getName() {
15    return this.name;
16  }
17
18  getFSurname() {
19    return this.fSurname;
20  }
21
22  getSSurname() {
23    return this.sSurname;
24  }
25
26  getFullName() {
27    return `${this.getName()} ${this.getFSurname()} ${this.getSSurname()}`;
28  }
29
30  getEmail() {
31    return this.email;
32  }
33
34  getRole() {
```

Figura 9.4: Imagen entidad User

A continuación, se muestra la imagen de la entidad **Teacher**, cabe destacar que en las entidades que heredan de **User**, antes de llamar a la clase padre para crear la entidad, se le asigna el rol que les pertenece, ya que uno de los requisitos para crear un usuario es que tenga un rol asignado.



```
1  'use strict';
2
3  const User = require('./User');
4  const { ROLES } = require('../constant');
5
6  class Teacher extends User {
7    constructor({ userData }) {
8      super({ userData: {
9        ...userData,
10        role: ROLES.TEACHER,
11      }});
12    }
13  }
14
15  module.exports = Teacher;
```

Figura 9.5: Imagen entidad Teacher

Otra de las entidades existentes es **Space**. Esta entidad representa un espacio en el que puede realizarse un examen, pudiendo distinguirse entre aula y laboratorio. Esta entidad, al igual que **User**, también dispone de los getters y setter para la consulta y modificación de los espacios de exámenes. Un espacio de examen se ha considerado que está representado por un id, un número de filas, un número de asientos y si se trata de un laboratorio o no.

The image shows a code editor window titled 'Space.js'. The code is written in JavaScript and defines a class for a 'Space' entity. It starts with 'use strict'; and then defines a class with a constructor and several methods. The constructor takes four arguments: id (with a default value of null), numRows, numSeats, and isLab. The methods include getId(), getRows(), getNumSeats(), isLab(), setRows(), and setSeats(). The code is as follows:

```
1 'use strict';
2
3 module.exports = class {
4   constructor(id : null := null, numRows, numSeats, isLab) {
5     this.id = id;
6     this.numRows = numRows;
7     this.numSeats = numSeats;
8     this.isLab = isLab;
9   }
10
11   getId() {
12     return this.id;
13   }
14
15   getRows() {
16     return this.numRows;
17   }
18
19   getNumSeats() {
20     return this.numSeats;
21   }
22
23   isLab() {
24     return this.isLab;
25   }
26
27   setRows(numRows) {
28     this.numRows = numRows;
29   }
30
31   setSeats(numSeats) {
32     this.numSeats = numSeats;
33   }
34 }
```

Figura 9.6: Imagen entidad Space

El archivo `Exam.js` es la entidad que representa un examen. Esta entidad está compuesta por un profesor, una asignatura, un espacio, una fecha, un grupo y un estado. El estado irá modificándose durante la ejecución del mismo y determinará aspectos como hasta cuando es posible que un alumno pueda entregar un examen o cuando es posible que un alumno pueda empezar un examen. `Exam.js` también dispone de métodos de getters y setters como puede ser el modificar la fecha del examen o consultar el profesor titular del examen.



```
1  const { STATE } = require('../constant');
2
3  class Exam {
4    constructor({ examData }) {
5      const { subject, group, teacher, date, space } = examData;
6
7      this.subject = subject;
8      this.group = group;
9      this.teacher = teacher;
10     this.space = space;
11     this.date = date;
12     this.state = STATE.ToDo;
13   }
14
15   getSubject() {
16     return this.subject;
17   }
18
19   getTeacher() {
20     return this.teacher;
21   }
22
23   getGroup() {
24     return this.group;
25   }
26
27   getDate() {
28     return this.date;
29   }
30
31   getSpace() {
```

Figura 9.7: Imagen entidad Exam

### 9.3.2. repositories

La otra carpeta existente en `Enterprise_business_rules` es `repositories`. Esta carpeta tiene como contenido los distintos repositorios que van a ser necesarios en la aplicación.

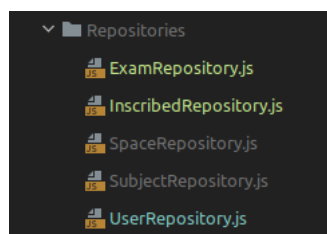


Figura 9.8: Imagen Clean Architecture

Un repositorio es una interfaz creada en la capa Enterprise Business Rules y que es

implementada por la capa Frameworks & drivers con el fin de conectar la parte de la lógica de la aplicación con la base de datos, de manera que no se rompa el principio de Clean Architecture.

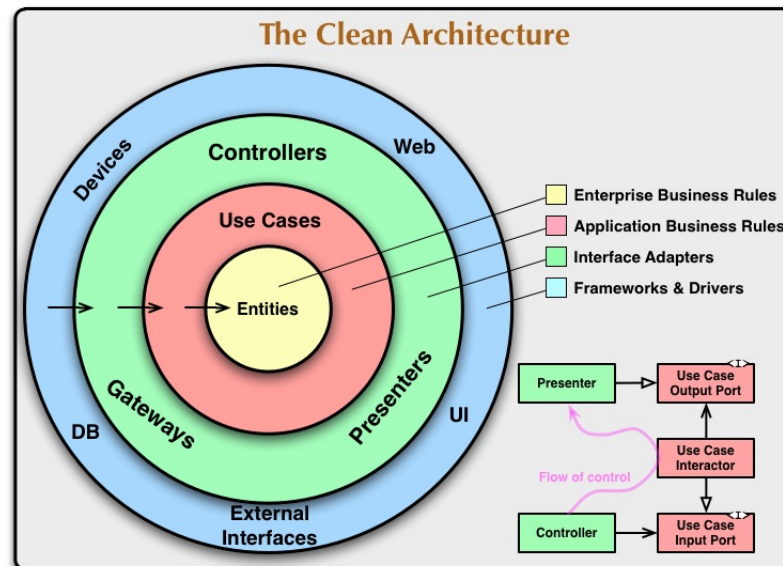


Figura 9.9: Imagen Clean Architecture

El principio básico de Clean Architecture es que las dependencias únicamente pueden ir hacia dentro. Pero en ocasiones, es necesario interactuar con la base de datos desde una capa interna, porque resulte necesario uno o varios datos para llevar a cabo un caso de uso. El caso es que, por el principio de dependencias de Clean Architecture, no es posible realizar una llamada a la base de datos desde un caso de uso, debido a que los casos de uso se encuentran en la capa Application Business Rules, y la base de datos se encuentra en la capa Frameworks & drivers, ahí es donde entran los repositorios. Un repositorio es una interfaz creada en la capa Enterprise Business Rules y que es implementada en la capa Frameworks & Drivers. Estos repositorios son los encargados de hacer la inversión de dependencias mencionada en el apartado Arquitectura, para no romper el principio de dependencias de Clean Architecture. Por lo tanto, cuando sea necesario algún dato, desde el caso de uso correspondiente, se llamará al repositorio que sea necesario, y obtendrá el dato sin romper el principio.

### 9.3.3. Gestión de errores

Toda aplicación, tarde o temprano, se encuentra con errores, ya sean causados por el usuario, como causados por el propio programador. Por ello, es importante tener una buena gestión de errores[12][15][41], porque aunque al principio es posible que no existan una gran cantidad de ellos, durante el periodo de desarrollo de la aplicación acabarán surgiendo. Es aquí donde entra la tercera carpeta de Enterprise\_business\_rules es Manage\_error, en la cual se ha alojado la implementación relacionada con la gestión de errores en la aplicación.

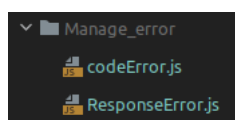
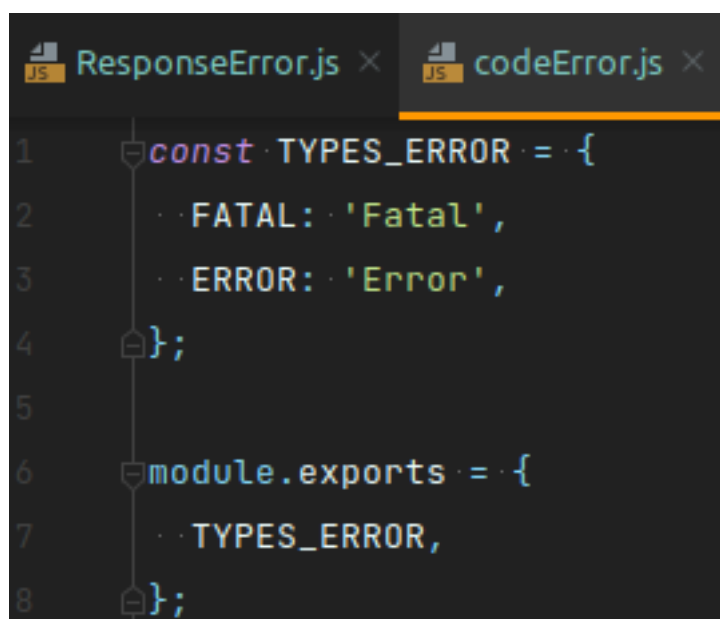


Figura 9.10: Imagen Manage error

En la imagen puede observarse que la carpeta contiene dos archivos, `codeError.js` y `ResponseError.js`.

Se va a comenzar por `codeError.js`. Este archivo sirve para almacenar los distintos tipos de errores que irán detectándose durante el desarrollo y posteriormente. Para ello, se añade como una constante a continuación de la última añadida, con el texto más adecuado para describir el error, y de esta manera, sea más sencilla y rápida la detección del error.

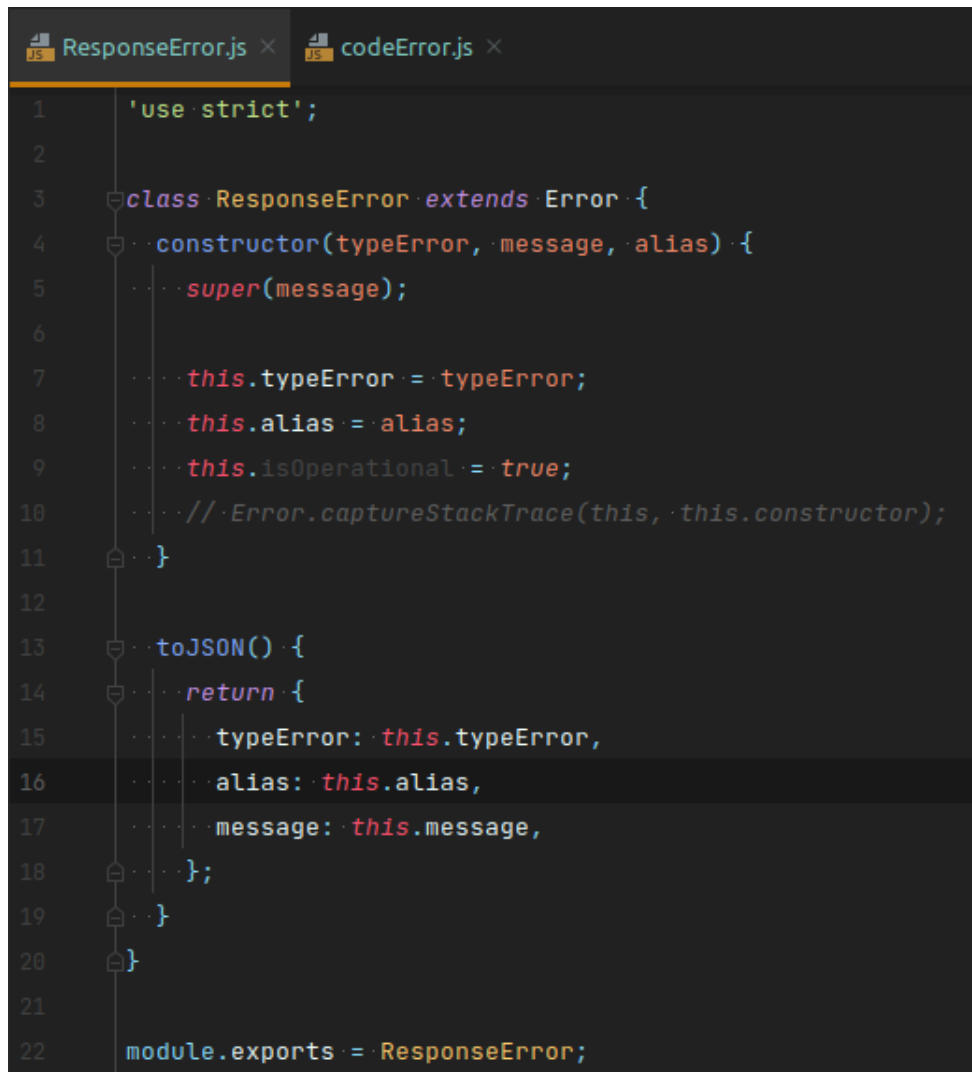
A screenshot of a code editor with two tabs open: 'ResponseError.js' and 'codeError.js'. The 'codeError.js' tab is active and shows the following JavaScript code:

```
1  const TYPES_ERROR = {  
2    FATAL: 'Fatal',  
3    ERROR: 'Error',  
4  };  
5  
6  module.exports = {  
7    TYPES_ERROR,  
8  };
```

Figura 9.11: Imagen codeError

El otro fichero es `ResponseError.js`. Este fichero es una extensión de la clase `Error`, que nos permite personalizar los errores manteniendo las ventajas de la clase padre.



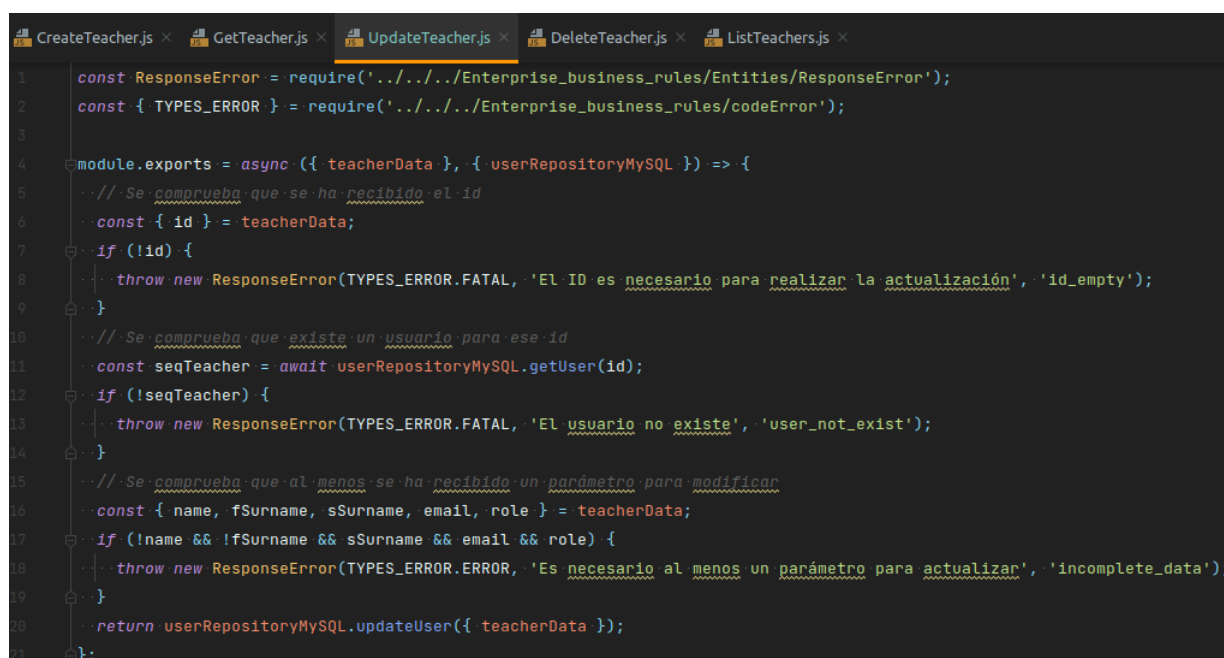


```
1  'use strict';
2
3  class ResponseError extends Error {
4    constructor(typeError, message, alias) {
5      super(message);
6
7      this.typeError = typeError;
8      this.alias = alias;
9      this.isOperational = true;
10     // Error.captureStackTrace(this, this.constructor);
11   }
12
13   toJSON() {
14     return {
15       typeError: this.typeError,
16       alias: this.alias,
17       message: this.message,
18     };
19   }
20 }
21
22 module.exports = ResponseError;
```

Figura 9.12: Imagen ResponseError

Este archivo está compuesto por un tipo de error, el cual está definido en `codeError.js`, un alias, y un mensaje. El mensaje, debe ser una frase indicando el error o su posible solución. Por otro lado los alias van a jugar un papel importante en la gestión de errores, ya que sería importante que tanto en el backend como en el frontend se compartieran los mismos alias con el fin de facilitar la detección y solución de errores, ya que éstos contendrán más información que únicamente los códigos de error.

Un ejemplo de utilización es el de el caso de uso de la actualización de un profesor.



```

1  const ResponseError = require('../../../Enterprise_business_rules/Entities/ResponseError');
2  const { TYPES_ERROR } = require('../../../Enterprise_business_rules/codeError');
3
4  module.exports = async ({ teacherData }, { userRepositoryMySQL }) => {
5    // Se comprueba que se ha recibido el id
6    const { id } = teacherData;
7
8    if (!id) {
9      throw new ResponseError(TYPES_ERROR.FATAL, 'El ID es necesario para realizar la actualización', 'id_empty');
10   }
11
12   // Se comprueba que existe un usuario para ese id
13   const seqTeacher = await userRepositoryMySQL.getUser(id);
14
15   if (!seqTeacher) {
16     throw new ResponseError(TYPES_ERROR.FATAL, 'El usuario no existe', 'user_not_exist');
17   }
18
19   // Se comprueba que al menos se ha recibido un parámetro para modificar
20   const { name, fSurname, sSurname, email, role } = teacherData;
21
22   if (!name && !fSurname && sSurname && email && role) {
23     throw new ResponseError(TYPES_ERROR.ERROR, 'Es necesario al menos un parámetro para actualizar', 'incomplete_data');
24   }
25
26   return userRepositoryMySQL.updateUser({ teacherData });
27 }

```

Figura 9.13: Imagen Use case UpdateTeacher

En la imagen, tanto en la línea 8, como en la 13 o en la 18, se puede apreciar cómo lanza el error y cómo el primer parámetro es el tipo de error, el segundo se trata del mensaje, y el tercero es el alias, el cual aporta más información sobre el error que se está reportando.

## 9.4. Application\_business\_rules

Basándonos en Clean Architecture, la capa Application\_bussiness\_rules es la encargada de albergar la lógica de la aplicación, así como la definición tanto de los datos de entrada, como los de salida y el comportamiento del sistema, todo ello en la carpeta Use\_cases.

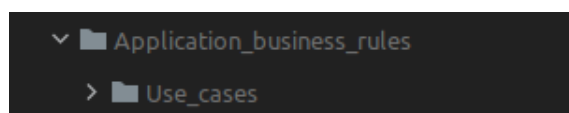


Figura 9.14: Imagen Application\_business\_rules

Esta carpeta, como su nombre indica, se encarga de almacenar todos los casos de uso, siendo éstos los encargados de la lógica de la aplicación.

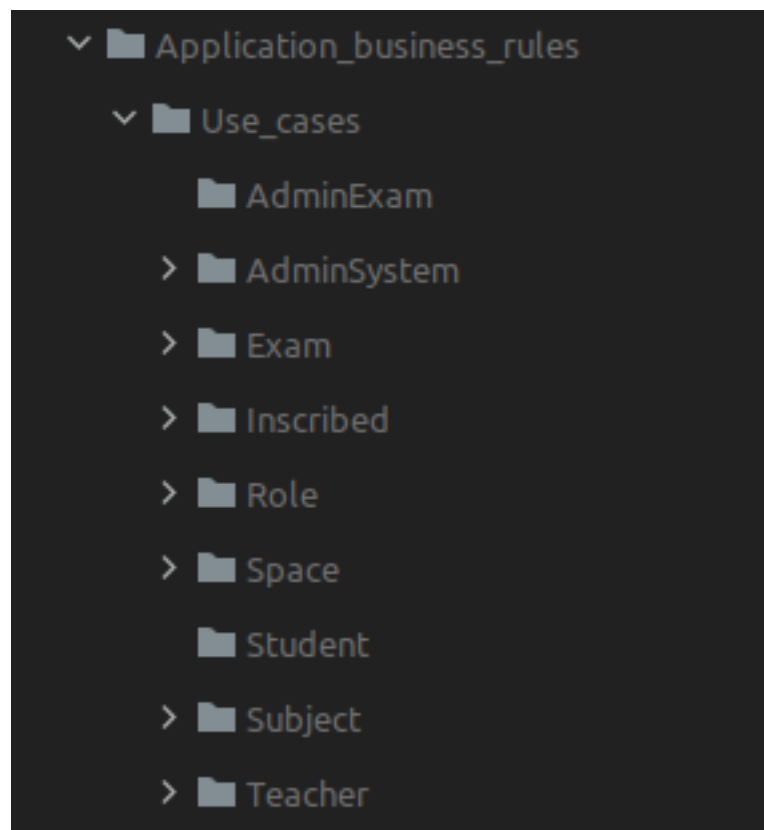


Figura 9.15: Imagen Use\_cases

Como se aprecia en la imagen, dentro de la carpeta Use\_cases, existen las carpetas AdminExam, AdminSystem, Exam, Space, Student, Teacher, Inscribed y Role. Se ha decidido organizar por entidad con el fin de estructurar los casos de uso de manera que la navegación entre ellos pueda ser rápida y sencilla.

Cada carpeta dentro de la carpeta Use\_cases se compone de los diferentes casos de uso para esa entidad, tratándose de una parte de la lógica de la aplicación.

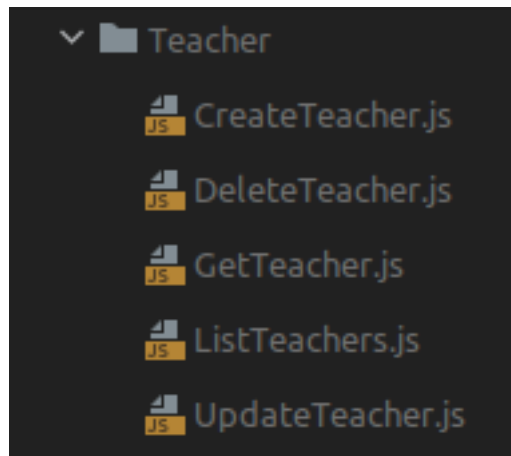


Figura 9.16: Imagen use case Teacher

En cada caso de uso, se debe comprobar que los datos de entrada requeridos son recibidos, y que además son correctos, de lo contrario, se lanzará una excepción con el error oportuno. A continuación, si es necesario realizar algún tipo de parseo de dato, se realiza en este momento. Y por último se llevan a cabo las acciones necesarias para satisfacer los requerimientos del caso de uso y devolver el resultado solicitado.

AdminExam, AdminSystem, Student, y Teacher son similares, ya que todos son roles de un usuario. Es por ello que únicamente voy a utilizar Teacher para el ejemplo, y como caso de uso ,UpdateTeacher.js.

```

1  const ResponseError = require('../../Enterprise_business_rules/Entities/ResponseError');
2  const { TYPES_ERROR } = require('../../Enterprise_business_rules/codeError');
3
4  module.exports = async ({ teacherData }, { userRepositoryMySQL }) => {
5    // Se comprueba que se ha recibido el id
6    const { id } = teacherData;
7    if (!id) {
8      throw new ResponseError(TYPES_ERROR.FATAL, 'El ID es necesario para realizar la actualización', 'id_empty');
9    }
10   // Se comprueba que existe un usuario para ese id
11   const seqTeacher = await userRepositoryMySQL.getUser(id);
12   if (!seqTeacher) {
13     throw new ResponseError(TYPES_ERROR.FATAL, 'El usuario no existe', 'user_not_exist');
14   }
15   // Se comprueba que al menos se ha recibido un parámetro para modificar
16   const { name, fSurname, sSurname, email, role } = teacherData;
17   if (!name && !fSurname && sSurname && email && role) {
18     throw new ResponseError(TYPES_ERROR.ERROR, 'Es necesario al menos un parámetro para actualizar', 'incomplete_data');
19   }
20   return userRepositoryMySQL.updateUser({ teacherData });
21 };

```

Figura 9.17: Imagen Update Teacher

Para el caso de actualizar un usuario los requerimientos son:

- Tener el id del usuario a actualizar. Éste id se obtiene mediante un dato de entrada por parámetro.
- Comprobación de que el usuario con el id que se ha recibido existe en la base de datos.
- Comprobación de que se ha recibido al menos un dato del usuario para ser actualizado.

Como se puede apreciar en la imagen, las tres primeras partes se dedican a comprobar datos y extraer datos que son necesarios para la obtención del resultado. En la primera parte hace la comprobación de que uno de los parámetros de entrada sea el id. El siguiente paso es la obtención del usuario a modificar mediante el id previamente verificado. Se realiza la petición a la base de datos y después se comprueba la existencia del usuario mediante el resultado obtenido de la consulta. A continuación se realiza la comprobación de que se ha recibido algún parámetro para ser actualizado. Por último, se manda la información al repositorio ubicado en la capa Enterprise Business Rules, el cual también es recibido por parámetro de entrada, para obtener una solución y devolverla.

## 9.5. Interface\_adapters

Interface adapters es la capa encargada de la transformación de datos, albergar los controladores y los accesos a servicios de terceros. Es por eso, que en esta parte se van a encontrar dos carpetas, Controller y Routes.

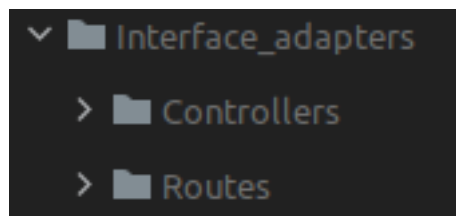


Figura 9.18: Imagen Interface\_adapters

A continuación, se va a pasar a explicar cada una de ellas.

### 9.5.1. Controllers

Los controladores son de gran importancia en cualquier arquitectura, ya que sin estos intermediarios entre la capa externa y la capa del core, cada vez que hubiera un cambio, por ejemplo, en la capa externa, sería necesario modificar también la capa del core, provocando que el tiempo de modificación sea más largo. En cambio, gracias a este intermediario llamado controlador, cuando algo en la capa externa debe cambiar, tan solo se tiene que adaptar el controlador a la capa externa, quedando la del core intacta. En esta aplicación,

se han creado los siguientes controladores.

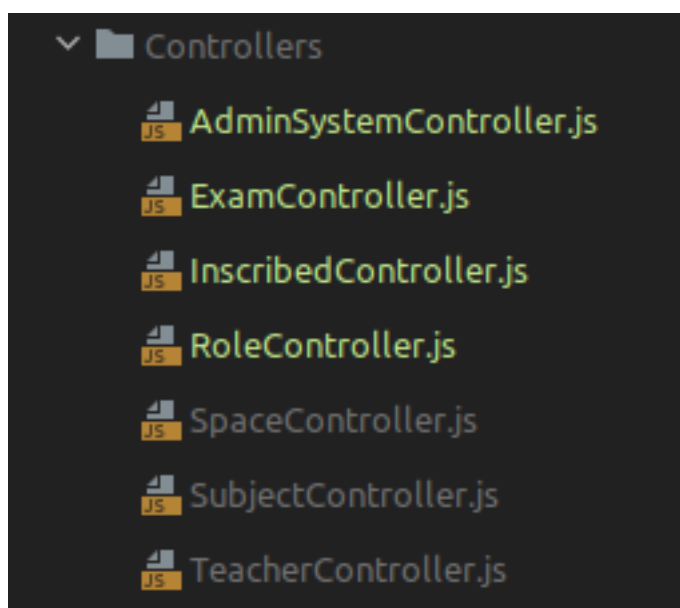
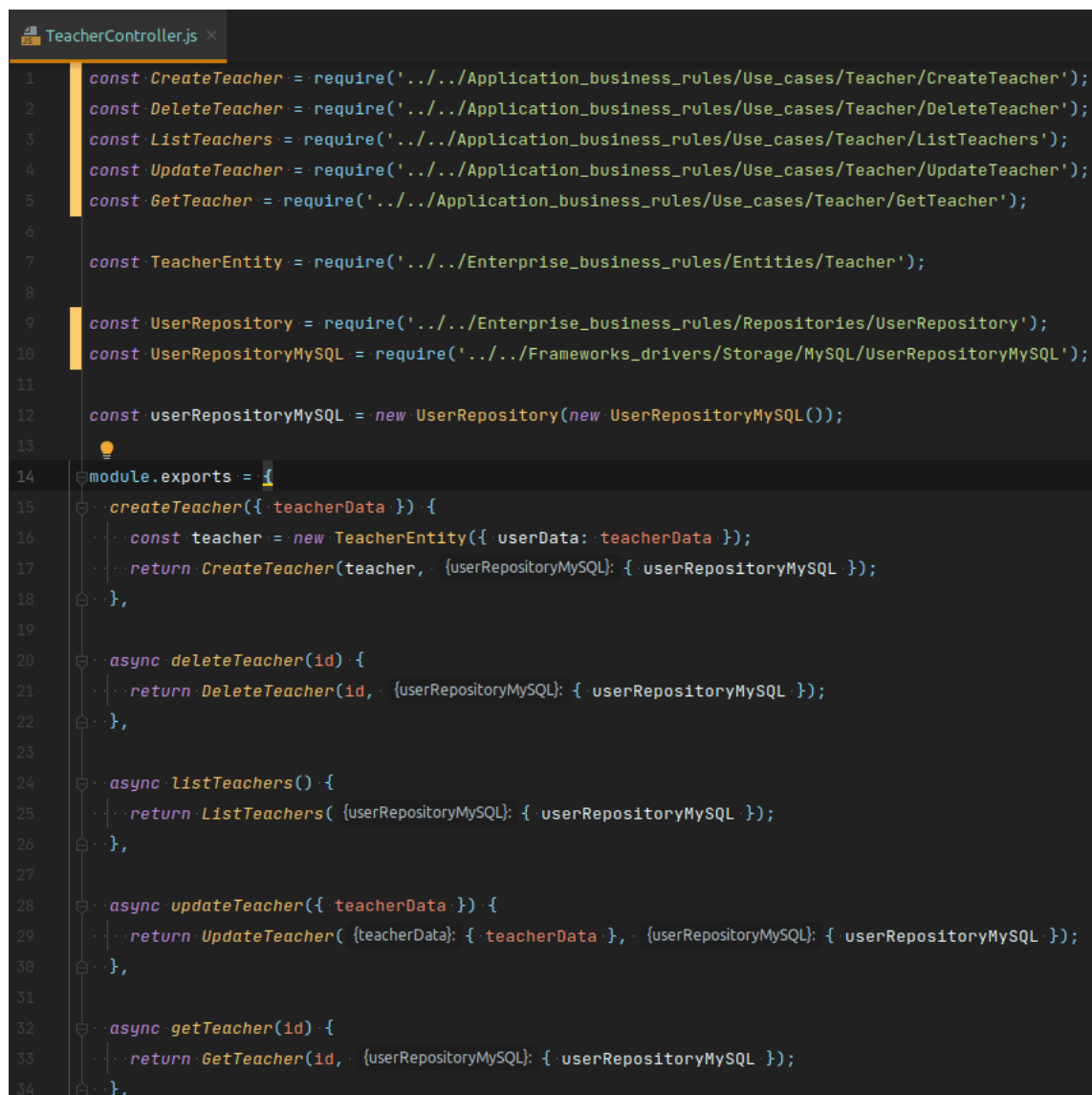


Figura 9.19: Imagen Controllers

Cada controlador se ocupa de unas funciones según su ámbito. Para que la idea quede más clara, se va a poner un ejemplo de ello con el controlador del profesor.



```

1  const CreateTeacher = require('../../Application_business_rules/Use_cases/Teacher/CreateTeacher');
2  const DeleteTeacher = require('../../Application_business_rules/Use_cases/Teacher/DeleteTeacher');
3  const ListTeachers = require('../../Application_business_rules/Use_cases/Teacher/ListTeachers');
4  const UpdateTeacher = require('../../Application_business_rules/Use_cases/Teacher/UpdateTeacher');
5  const GetTeacher = require('../../Application_business_rules/Use_cases/Teacher/GetTeacher');
6
7  const TeacherEntity = require('../../Enterprise_business_rules/Entities/Teacher');
8
9  const UserRepository = require('../../Enterprise_business_rules/Repositories/UserRepository');
10 const UserRepositoryMySQL = require('../../Frameworks_drivers/Storage/MySQL/UserRepositoryMySQL');
11
12 const userRepositoryMySQL = new UserRepository(new UserRepositoryMySQL());
13
14 module.exports = {
15   createTeacher({ teacherData }) {
16     const teacher = new TeacherEntity({ userData: teacherData });
17     return CreateTeacher(teacher, { userRepositoryMySQL: { userRepositoryMySQL } });
18   },
19
20   async deleteTeacher(id) {
21     return DeleteTeacher(id, { userRepositoryMySQL: { userRepositoryMySQL } });
22   },
23
24   async listTeachers() {
25     return ListTeachers({ userRepositoryMySQL: { userRepositoryMySQL } });
26   },
27
28   async updateTeacher({ teacherData }) {
29     return UpdateTeacher({ teacherData: { teacherData }, { userRepositoryMySQL: { userRepositoryMySQL } });
30   },
31
32   async getTeacher(id) {
33     return GetTeacher(id, { userRepositoryMySQL: { userRepositoryMySQL } });
34   },

```

Figura 9.20: Imagen TeacherController

Este controlador se ocupará de las acciones relacionadas con un profesor, como puede ser crear un profesor, eliminar un profesor o actualizar un profesor. Aunque como se ve en la imagen en la línea 12, se crea una instancia del repositorio, esto no es correcto, dado que desde la capa Interface adapters no se debe acceder a la capa Frameworks & drivers. Lo que se debe hacer es crear un archivo de configuración, generalmente llamado `env`, en el que se configuran las variables de entorno, es aquí donde se le especificará a la aplicación que base de datos es la que se va a desear usar. Una vez se tenga ese archivo, se creará una instancia del repositorio que sea necesario en el archivo y posteriormente será importado desde donde actualmente se crea la instancia.

Una vez se disponga de la instancia necesaria para continuar, se llama a la función solicitada y se le pasan los datos requeridos y el repositorio.

### 9.5.2. Routes

En esta carpeta se almacena la API. Tener una API aporta multitud de ventajas como es la separación entre cliente y servidor, escalabilidad o independencia del tipo de plataforma. En el caso de esta aplicación, aporta exactamente lo mismo. Al tener la API, existe una clara diferencia entre la parte del frontend, y la parte del backend, comunicadas ambas por la API que permite cambiar la parte frontend sin necesidad de cambiar el backend. Aporta escalabilidad, ya que al tener separados el cliente y servidor, es posible añadir nuevas funcionalidades sin grandes complicaciones. Y aporta independencia del tipo de plataforma, es posible tener servidores ya sean Java, python o Node.js, siempre cuando el intercambio de datos sea en el formato acordado, en el caso de este proyecto, JSON.

La carpeta del proyecto está compuesta por los siguientes archivos:

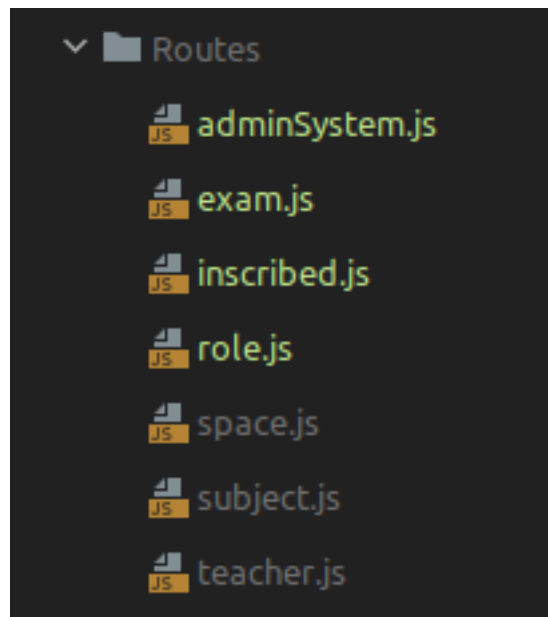
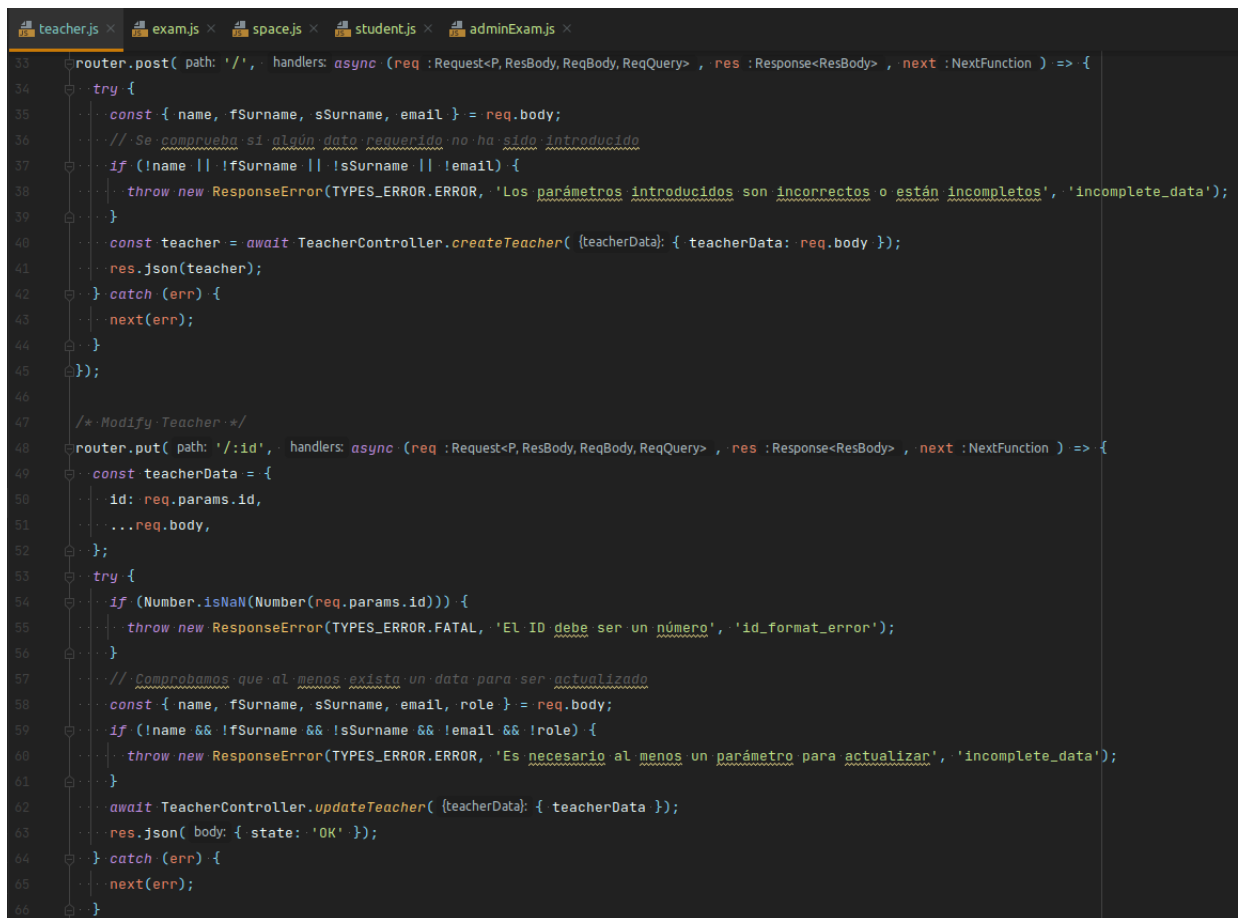


Figura 9.21: Imagen Routes

Para ver un ejemplo del funcionamiento de ello, vamos a ver las rutas de `teacher.js`.





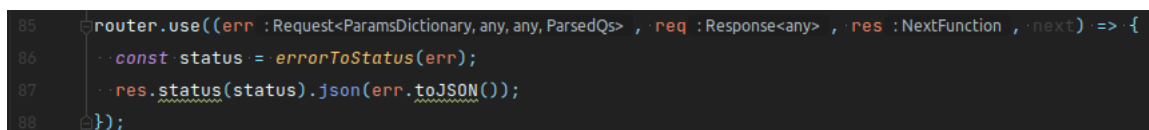
```

43 router.post( path: '/', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery> , res : Response<ResBody> , next : NextFunction ) => {
44   try {
45     const { name, fSurname, sSurname, email } = req.body;
46     // Se comprueba si algún dato requerido no ha sido introducido
47     if (!name || !fSurname || !sSurname || !email) {
48       throw new ResponseError(TYPES_ERROR.ERROR, 'Los parámetros introducidos son incorrectos o están incompletos', 'incomplete_data');
49     }
50     const teacher = await TeacherController.createTeacher( {teacherData: { teacherData: req.body }});
51     res.json(teacher);
52   } catch (err) {
53     next(err);
54   }
55 });
56
57 /* Modify Teacher */
58 router.put( path: '/:id', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery> , res : Response<ResBody> , next : NextFunction ) => {
59   const teacherData = {
60     id: req.params.id,
61     ...req.body,
62   };
63   try {
64     if (Number.isNaN(Number(req.params.id))) {
65       throw new ResponseError(TYPES_ERROR.FATAL, 'El ID debe ser un número', 'id_format_error');
66     }
67     // Comprobamos que al menos exista un dato para ser actualizado
68     const { name, fSurname, sSurname, email, role } = req.body;
69     if (!name && !fSurname && !sSurname && !email && !role) {
70       throw new ResponseError(TYPES_ERROR.ERROR, 'Es necesario al menos un parámetro para actualizar', 'incomplete_data');
71     }
72     await TeacherController.updateTeacher( {teacherData: { teacherData }});
73     res.json( body: { state: 'OK' } );
74   } catch (err) {
75     next(err);
76   }
77 }

```

Figura 9.22: Imagen route teacher

En la imagen se muestra la implementación de los endpoints referentes a la creación de un profesor, utilizando el método `post`, y a la actualización de un profesor, utilizando el método `put`. En ambos métodos se aprecia que el primer paso es la validación de los datos recibidos, y que en caso de no ser correctos o no haberlos recibido, se lanza una excepción con el código de error, el mensaje y el alias correspondiente. A continuación, ambos métodos realizan una llamada al método correspondiente del controlador pasándole por parámetro los datos necesarios. Después, devolverá un JSON con la solución o información a la solicitud. Si durante el proceso de la solicitud hubiera algún problema que provocase algún tipo de error, será capturado por el `catch`, y con la función `next()`, una función de Express que provoca que se pase al siguiente endpoint en los cuales no entrará debido a que la ruta no coincide con la introducida por el usuario, hasta llegar a la siguiente:



```

85 router.use((err : Request<ParamsDictionary, any, any, ParsedQs> , req : Response<any> , res : NextFunction , next) => {
86   const status = errorToStatus(err);
87   res.status(status).json(err.toJSON());
88 });

```

Figura 9.23: Imagen route teacher

Es un middleware de Express que se encarga de manejar los errores y que está compuesto por cuatro parámetros de entrada, `err`, `req`, `res` y `next`. Esta función debe ponerse al final de todos los endpoints, y en el caso de utilizar otro `router.use()`, debe estar después, osea, al final.

## 9.6. Frameworks\_drivers

Frameworks & Drivers es la capa exterior de Clean Architecture, la llamada capa de los “detalles”, puesto que es en esta capa donde se encuentra la implementación de la base de datos, los detalles de la configuración, desde donde se conecta con servicios de terceros, donde está la web. Es lógico que esto se encuentre en la capa exterior ya que la intención de Clean Architecture es que cuanto más interior sea la capa, menos implementación se encuentre, con el fin de reducir el impacto de una modificación en la aplicación.

En esta aplicación, en esta capa existen cuatro carpetas, Database, ORM, Storage y WebServer.

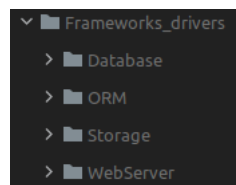


Figura 9.24: Imagen Frameworks\_drivers

A continuación pasamos a ver cada carpeta y su contenido.

### 9.6.1. Database

En esta carpeta se almacenan los modelos de sequelize en relación a las tablas de la base de datos.

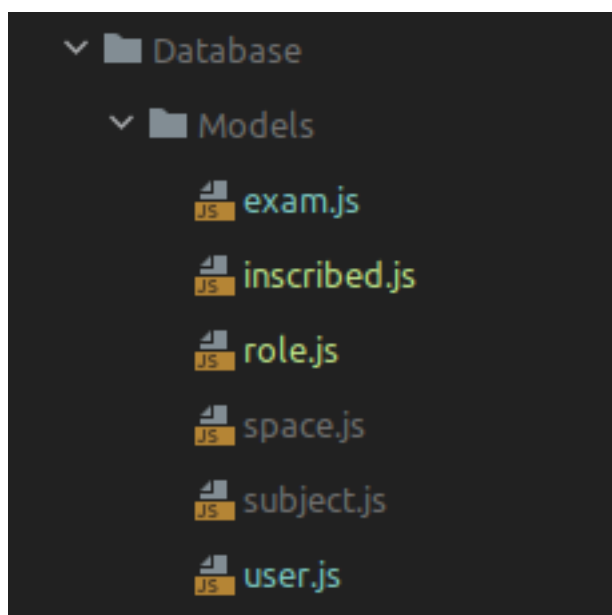
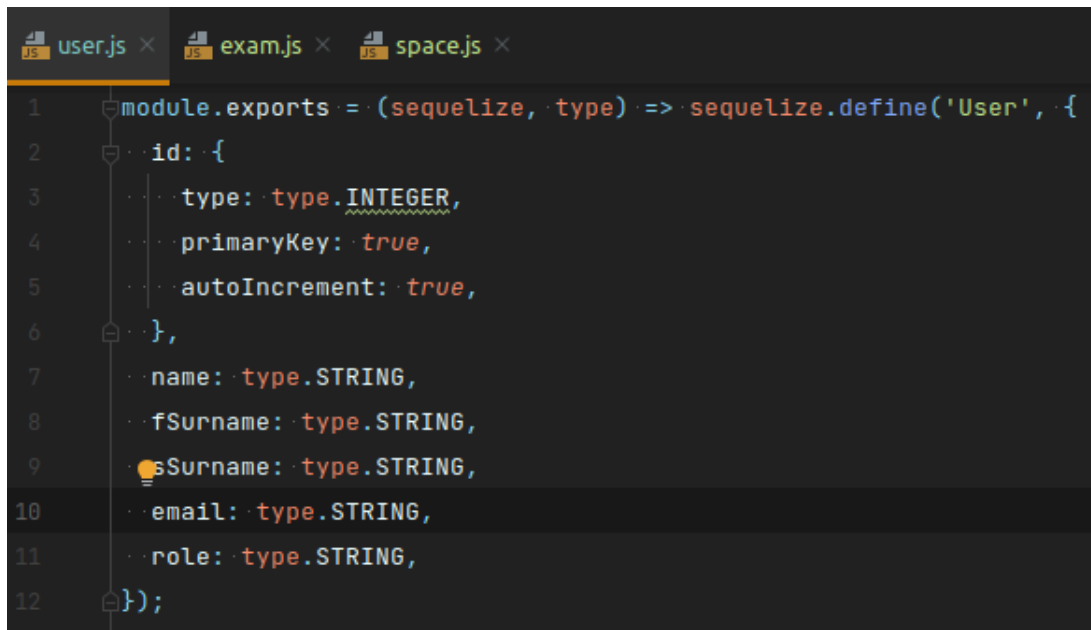


Figura 9.25: Imagen Database

En estos modelos se define el nombre de la tabla y los campos de la tabla, poniendo el nombre del campo y su tipo. Sequelize también ofrece la posibilidad de añadir restricciones o condiciones a los campos, ya que en el caso de ejecutar la aplicación, y detectar que la tabla no existe, se crea automáticamente con el modelo creado y las restricciones o condiciones para ello.

Para mayor claridad, se va a exponer un ejemplo con el modelo `user.js`.



```
1 module.exports = (sequelize, type) => sequelize.define('User', {
2   id: {
3     type: type.INTEGER,
4     primaryKey: true,
5     autoIncrement: true,
6   },
7   name: type.STRING,
8   fSurname: type.STRING,
9   sSurname: type.STRING,
10  email: type.STRING,
11  role: type.STRING,
12 });
```

Figura 9.26: Imagen Modelo user

Para crear un modelo de Sequelize se utiliza una función llamada **define**, esta función contiene dos parámetros de entrada, el primero es el nombre que tiene, o en el caso de no existir, tendrá la tabla que va a ser creada. El segundo parámetro es la definición de los campos de la tabla. Como se anticipaba al principio, está compuesto de un nombre, el del campo, y un tipo, como en este caso es un usuario, éste estará compuesto por un nombre, un primer apellido, un segundo apellido, un email, un rol y un id. Como se aprecia en la imagen, es en el id donde se aprecian los detalles previamente mencionados, ya que ha sido a éste al que se le ha añadido que sea clave primaria, y que además de eso, cada vez que se añada un usuario a la tabla, el id se autoincremente.

### 9.6.2. ORM

Para la realización de la parte de sequelize, se consultó una serie de tutoriales[21][22][23][24] que llevaban a cabo el proceso desde cero. En esta carpeta se aloja todo lo relacionado con la configuración de Sequelize, es por eso que se encuentra el archivo **sequelize.js**. Aunque los modelos y la configuración de sequelize está separada, si en un futuro se decidiese a incluir otro ORM, los modelos creados con Sequelize podrían reubicarse en una carpeta creada dentro de ORM llamada Sequelize, donde estarían tanto los modelos creados por Sequelize, como la configuración.

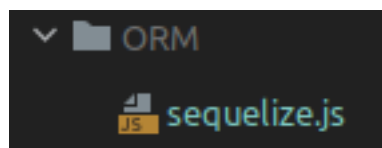
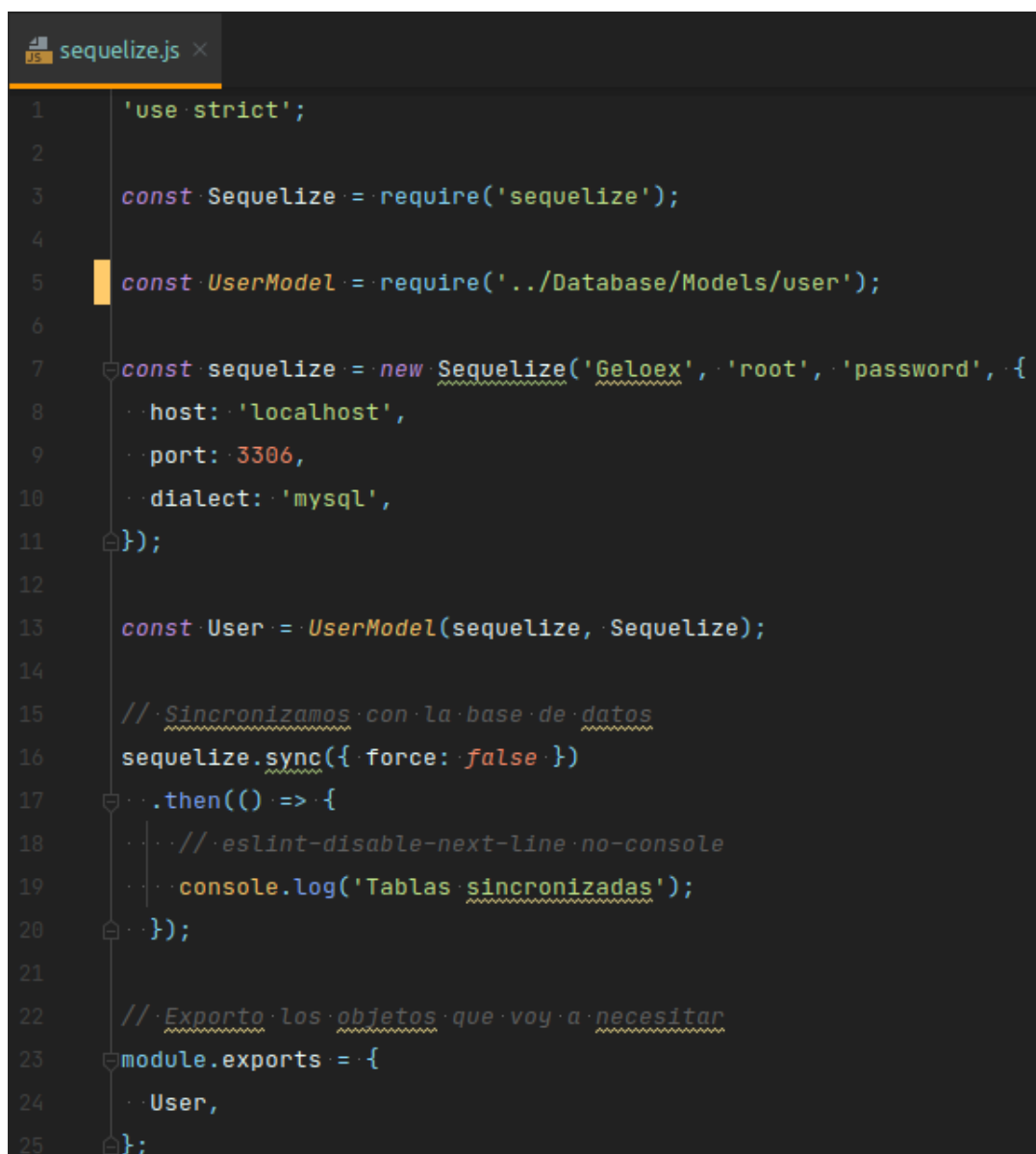


Figura 9.27: Imagen ORM

Este archivo contiene la configuración de la base de datos, que es el sitio en el que se crea una instancia de Sequelize con cuatro argumentos. El primero, el nombre de la base de datos, el segundo es el usuario de la base de datos, el tercero es la contraseña de la base de datos para ese usuario y el cuarto es una serie de campos. Estos campos indican el host, el puerto y el tipo de base de datos que se utiliza. Lo siguiente en el archivo es crear una instancia del modelo, para poder exportarla mediante el `module.exports` de javascript, lo cual permite desde cual punto de la aplicación, importar el modelo. Por último se ha realizado la sincronización con las tablas mediante el método `sync` de sequelize, que, dependiendo de si `force` es `true` o `false`, te borra toda la base de datos o la deja tal cual está. Esto resulta realmente útil cuando se están probando funcionalidades y no se quiere acabar con la tabla llena, o que la tabla empiece con un `id=200`, por ejemplo.



```
1 'use strict';
2
3 const Sequelize = require('sequelize');
4
5 const UserModel = require('../Database/Models/user');
6
7 const sequelize = new Sequelize('Geloex', 'root', 'password', {
8   host: 'localhost',
9   port: 3306,
10  dialect: 'mysql',
11 });
12
13 const User = UserModel(sequelize, Sequelize);
14
15 // Sincronizamos con la base de datos
16 sequelize.sync({ force: false })
17   .then(() => {
18     // eslint-disable-next-line no-console
19     console.log('Tablas sincronizadas');
20   });
21
22 // Exporto los objetos que voy a necesitar
23 module.exports = {
24   User,
25 }
```

Figura 9.28: Imagen Sequelize

### 9.6.3. Storage

Esta carpeta es la encargada de alojar las implementaciones de las interfaces de los repositorios creadas en la capa Enterprise Business Rules y que son usadas por los casos de uso para interactuar con la base de datos mediante la inversión de dependencias ya mencionada. Como es aquí donde se almacenarían todo los repositorios para cualquier base de datos, se ha optado por crear una carpeta llamada MySQL, donde alojar los repositorios relacionados con MySQL.

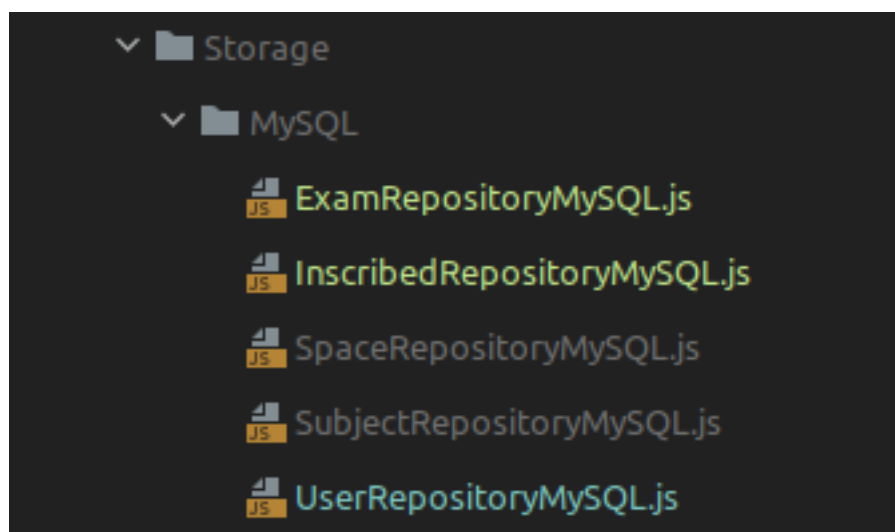
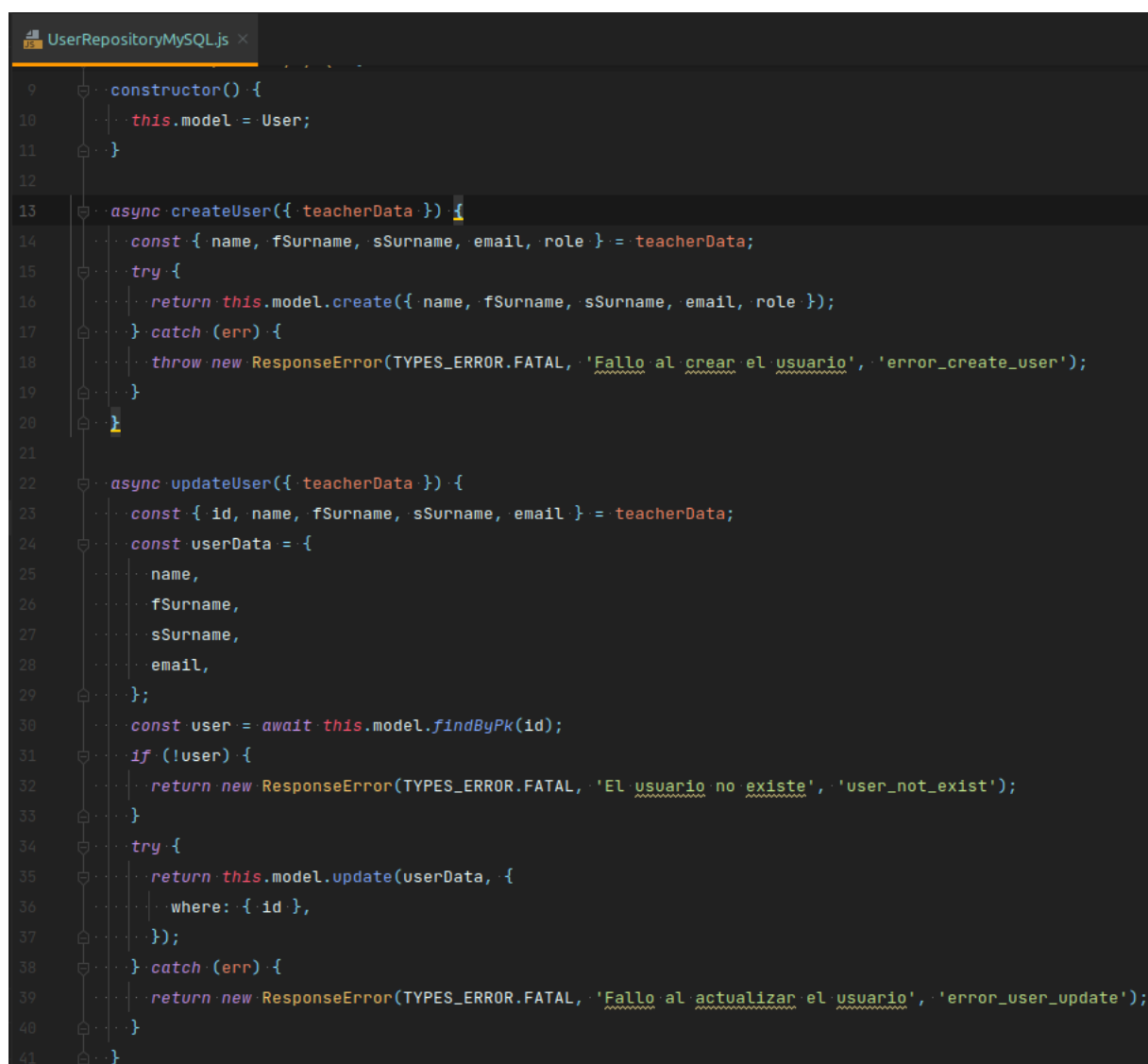


Figura 9.29: Imagen Storage

A continuación se va a mostrar un ejemplo de implementación de un repositorio, concretamente se mostrará el de usuarios, `UserRepositoryMySQL`.



```
9 constructor() {-{
10   this.model = User;
11 }
12
13 async createUser({ teacherData }) {-{
14   const { name, fSurname, sSurname, email, role } = teacherData;
15   try {-{
16     return this.model.create({ name, fSurname, sSurname, email, role });
17   } catch (err) {-{
18     throw new ResponseError(TYPES_ERROR.FATAL, 'Fallo al crear el usuario', 'error_create_user');
19   }
20 }
21
22 async updateUser({ teacherData }) {-{
23   const { id, name, fSurname, sSurname, email } = teacherData;
24   const userData = {-{
25     name,
26     fSurname,
27     sSurname,
28     email,
29   };
30   const user = await this.model.findOne({ id });
31   if (!user) {-{
32     return new ResponseError(TYPES_ERROR.FATAL, 'El usuario no existe', 'user_not_exist');
33   }
34   try {-{
35     return this.model.update(userData, {-{
36       where: { id },
37     });
38   } catch (err) {-{
39     return new ResponseError(TYPES_ERROR.FATAL, 'Fallo al actualizar el usuario', 'error_user_update');
40   }
41 }
```

Figura 9.30: Imagen UserRepositoryMySQL

Lo primero que es necesario para implementar el repositorio es importar el modelo que se ha creado con sequelize en el modelo `user.js` alojado en `Database/Models`. Ese modelo es el que se utiliza para interactuar con la base de datos, con funciones como `create`, `destroy`, `findOne`, `findAll`, entre otras.

En la imagen se puede observar que los métodos son asíncronos, esto resulta ser así debido a que Sequelize, como resultado, devuelve una promesa en javascript. Para tratar las promesas en Javascript, se ha utilizado `async` y `await`. El `await` se ha colocado donde se recepciona la promesa, que se encuentra en la API, en el endpoint desde el que se ha iniciado la llamada hasta llegar `UserRepositoryMySQL`. Utilizar `async` y `await` es relativamente nuevo en Javascript, es por eso que también es posible realizarlo con `new Promise` y recepcionarlo con `.then`.



### 9.6.4. WebServer

Esta carpeta contiene todo lo relacionado con el servidor. Actualmente, únicamente contiene un único archivo, `server.js`.



Figura 9.31: Imagen WebServer

En este archivo se configura todo lo que implica al servidor. También es donde se ubican todas las rutas de la API, debiendo estar todas las rutas creadas añadidas en este archivo.

La primera parte, en la que se crea una constante llamada `UserValidation`, es una función creada gracias a la librería `Joi`, la cual viene incluida en `express-validation`, y que se utiliza para validación de datos. En este caso, es una validación de usuario, ya que un usuario debe estar formado por un nombre, apellidos, email, que tenga una contraseña y tenga asignado un id, en el caso de no ser uno nuevo. Este tipo de validación resulta ser realmente útil ya que puede ser usada para validar un profesor, un alumno, un administrador de sistemas o un administrador de exámenes, y únicamente es necesario hacer lo que se aprecia en la siguiente imagen. Cuando se incluya un conjunto de rutas, como por ejemplo las del profesor, `teacherRouter`, se añade como parámetro para validar. Además, es posible utilizarla en más de un conjunto de rutas, de esta manera reutilizamos código y no se escribe el mismo código más de una vez.

Como ya se anticipaba, la segunda parte es la zona en la que se añaden los distintos conjuntos de rutas a la aplicación. Para ello, se utiliza el middleware de Express, concretamente la función `use`. En la línea 41 es posible observar como son añadidas las rutas referentes a un profesor, y cómo es utilizado el método `use` para ello. Con el fin de evitar el fallo de la aplicación por no haber incluido correctamente una ruta, se utiliza el método `all`. Este método se debe situar en último lugar, ya que, siempre que se llegue aquí, se ejecuta. Por lo tanto, este método será utilizado con el fin de que si una ruta no está contemplada en la aplicación, llegue a ese método y notifique que la ruta no se ha encontrado.

La última parte está compuesta por la ejecución de la aplicación, en ella, se utilizará el método `listen`, al cual hay que introducirle dos argumentos, el puerto, y un callback, en caso de querer hacer algo cuando la aplicación haya arrancado. En el caso de esta aplicación, se ha escrito una frase para notificar que la aplicación ha sido arrancada y el puerto donde lo ha hecho.

```

server.js x
22
23 const UserValidation = {
24   body: Joi.object({
25     name: Joi.string()
26       .required(),
27     fSurname: Joi.string()
28       .required(),
29     sSurname: Joi.string()
30       .required(),
31     email: Joi.string()
32       .email()
33       .required(),
34     password: Joi.string()
35       .regex( str: /[a-zA-Z0-9]{3,30}/),
36     id: Joi.number()
37       .integer(),
38   }),
39 };
40
41 app.use('/teacher', validate(UserValidation, { options: {}, joiRoot: {} }, teacherRouter));
42 app.all( path: '*', handlers: (req :Request<P, ResBody, ReqBody, ReqQuery> , res :Response<ResBody> , next :NextFunction ) => {
43   next(new ResponseError('No se puede encontrar ${req.originalUrl} en este servidor', 404));
44 });
45
46 // Para arrancar el servidor: listen(puerto, callback)
47 const server = app.listen( port: 3000, callback () => {
48   const { port } = server.address();
49   // eslint-disable-next-line no-console
50   console.log('*** App arrancada en el puerto:', port);
51 });

```

Figura 9.32: Imagen server

## 9.7. Escenario de uso

En esta sección se va a dar paso a explicar el flujo de la aplicación desde que le llega una petición, hasta su finalización. Para ello, se va a utilizar como ejemplo el caso de uso de crear un profesor. El diagrama de una funcionalidad que lleva la aplicación es el siguiente.

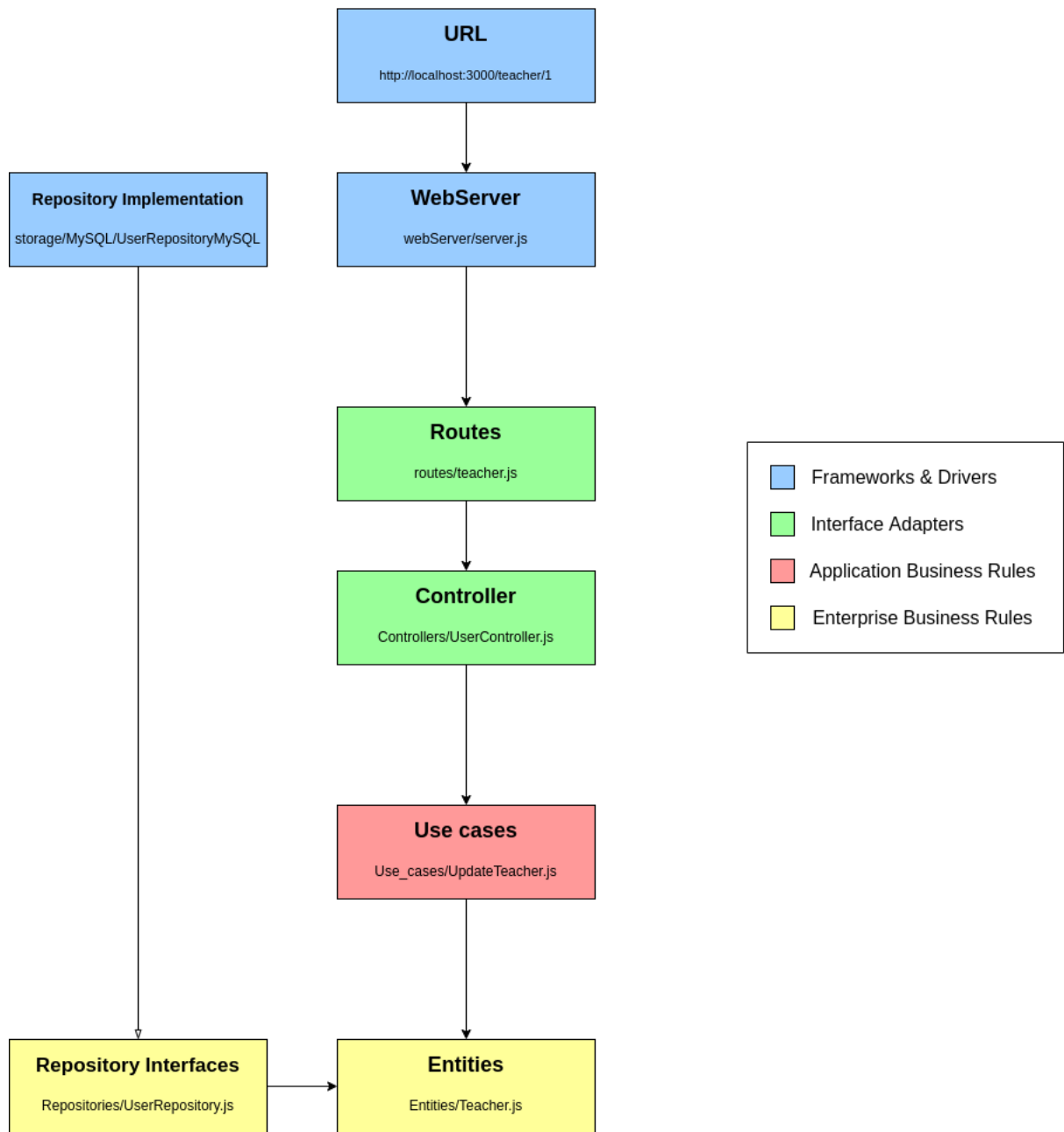


Figura 9.33: Imagen server

En primer lugar, se comprueba que la tabla **User** no tiene usuarios. Para ello se ha utilizado el entorno DataGrip, un IDE de la familia JetBrains que facilita la consulta de tablas, ya que sin comandos, es posible verlo de manera visual tan solo haciendo click en la tabla.

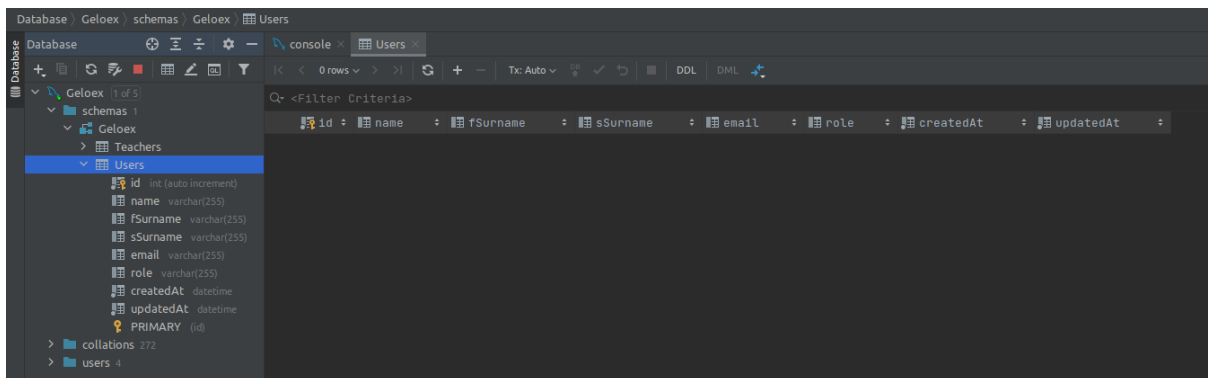


Figura 9.34: Imagen Tabla User vacía

Una vez comprobado que no existe ningún usuario registrado en la tabla **User**. Se proceder a mandar la solicitud de petición de creación de profesor. Para realizar las solicitudes a la aplicación, se ha utilizado postman, una aplicación ampliamente extendida para la comprobación de funcionalidades. Como lo que se desea es crear un nuevo profesor, seleccionamos el método POST, y a continuación, escribimos la URL de la petición. Como lo que se desea crear es un profesor debe comenzar por `http://localhost:3000/teacher`, que resulta ser la ruta de creación de profesores. A continuación, se rellena los argumentos que son necesarios para la creación de un profesor, nombre, apellidos y email.

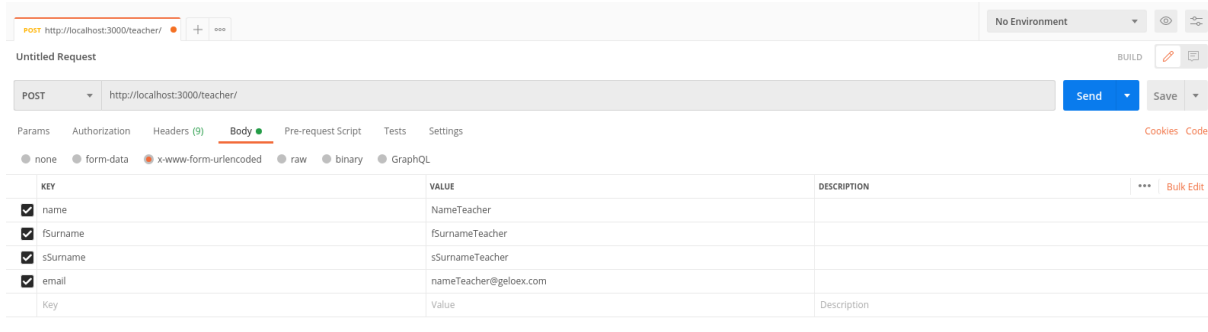
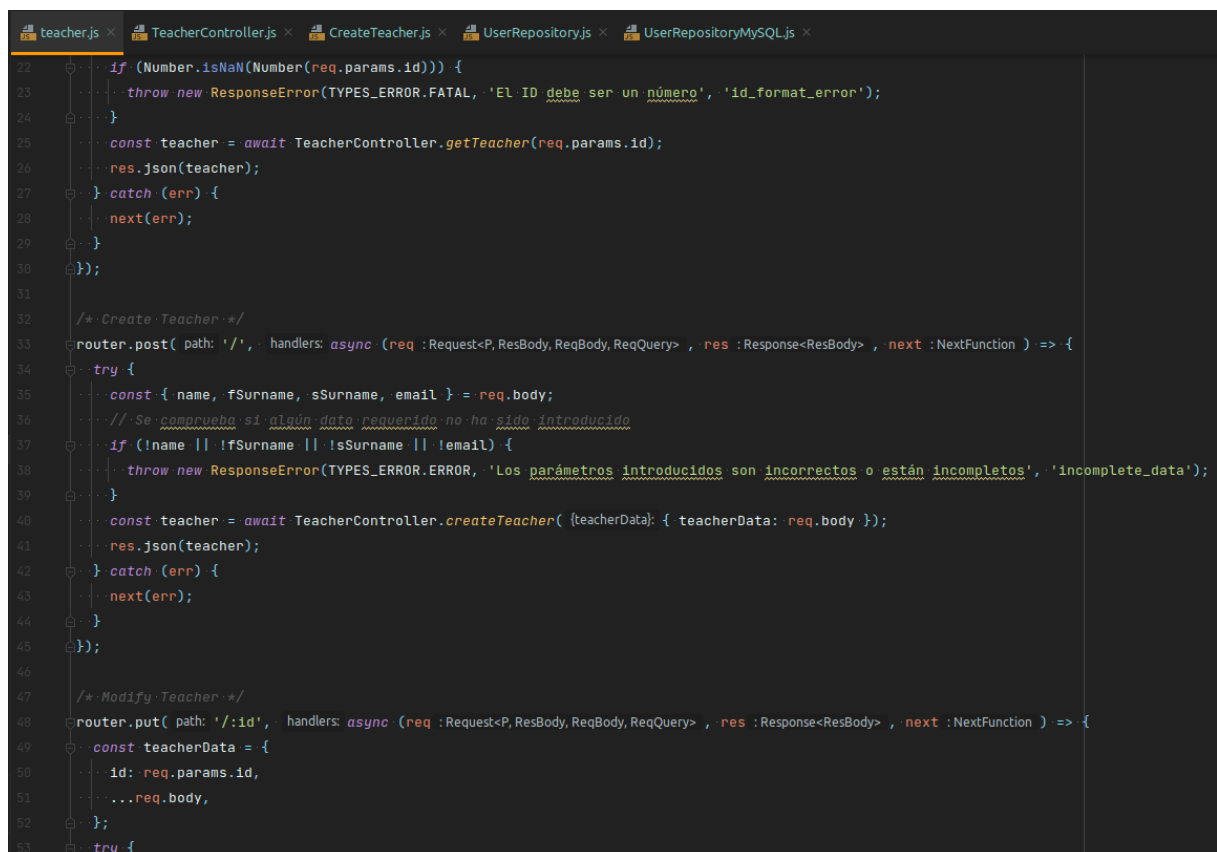


Figura 9.35: Imagen petición de creación de profesor en Postman

A continuación, se envía la petición desde Postman y pasaría al apartado de las rutas del profesor, situado en `Inteface_adapters/Routes/teacher.js`. De las rutas existentes en este fichero, entraría en el que utiliza el método POST, situado en la línea 33. Según entre en la petición, comprueba los datos **name**, **fSurname**, **sSurname** y **email** que ha recibido, y si falta algún dato, lanza un error. Después realiza la llamada al método de creación de profesor, **createTeacher**, que existe en el controlador del profesor, **TeacherController**. Nótese que en esta línea, en la llamada al controlador, se introduce **await**. Esto es necesario porque en la petición a la base de datos se utilizan promesas de Javascript, y con el **await** se le está diciendo a la ejecución, dado que Javascript es un lenguaje asíncrono, que no continúe la ejecución hasta que esta promesa no esté resuelta. Este fue uno de los problemas que se han tenido durante el desarrollo, el manejo de la asincronía en Javascript, ya

que por desconocimiento de ello, como no se había hecho la recepción correctamente de la promesa, el código continuaba y no aparecía solución porque la promesa había quedado en estado pending.



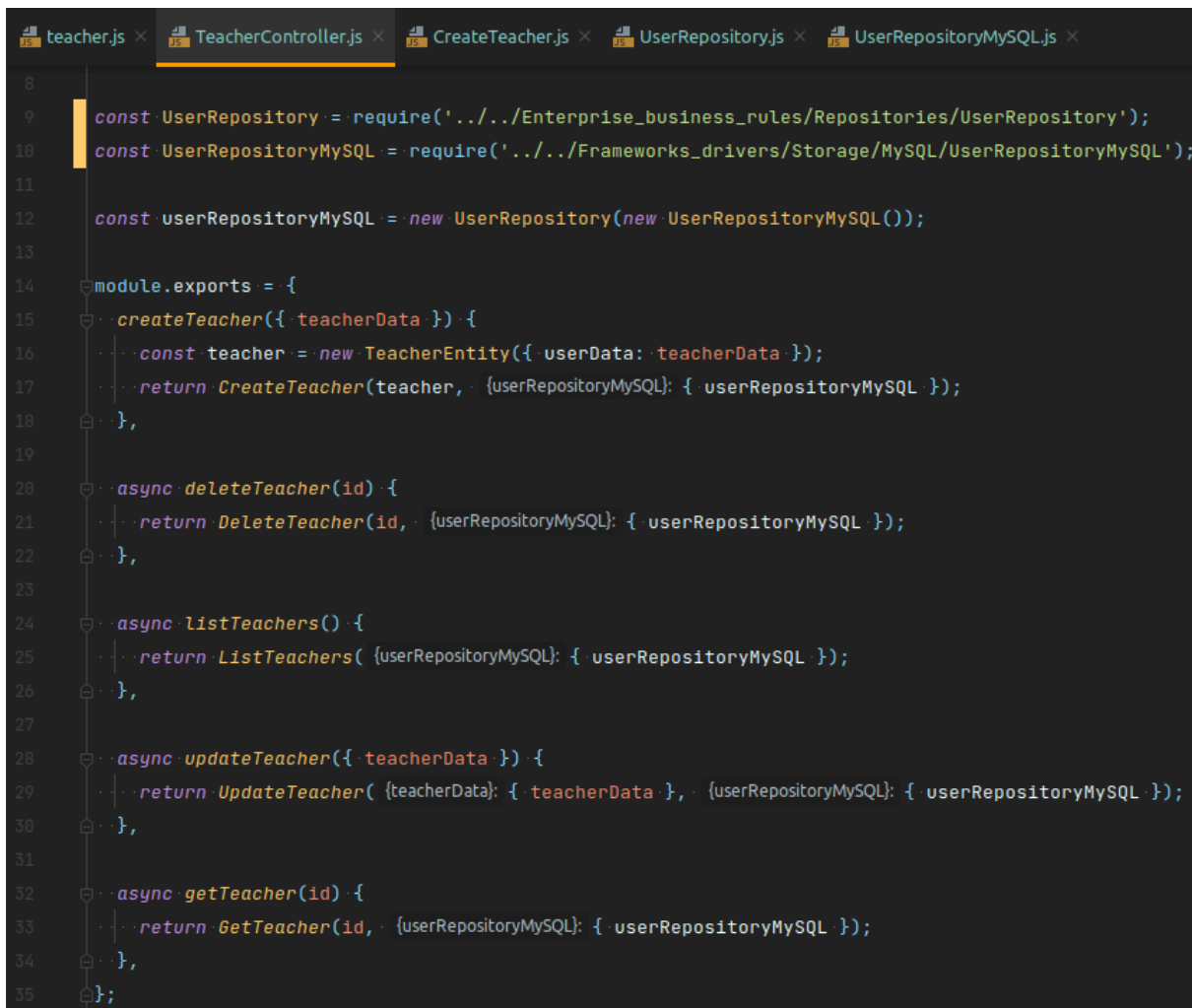
```

22  if (Number.isNaN(Number(req.params.id))) {
23    throw new ResponseError(TYPES_ERROR.FATAL, 'El ID debe ser un número', 'id_format_error');
24  }
25  const teacher = await TeacherController.getTeacher(req.params.id);
26  res.json(teacher);
27 } catch (err) {
28   next(err);
29 }
30 });
31
32 /* Create Teacher */
33 router.post( path: '/', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery> , res : Response<ResBody> , next : NextFunction ) => {
34   try {
35     const { name, fSurname, sSurname, email } = req.body;
36     // Se comprueba si algún dato requerido no ha sido introducido
37     if (!name || !fSurname || !sSurname || !email) {
38       throw new ResponseError(TYPES_ERROR.ERROR, 'Los parámetros introducidos son incorrectos o están incompletos', 'incomplete_data');
39     }
40     const teacher = await TeacherController.createTeacher( {teacherData: { teacherData: req.body }});
41     res.json(teacher);
42   } catch (err) {
43     next(err);
44   }
45 });
46
47 /* Modify Teacher */
48 router.put( path:('/:id)', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery> , res : Response<ResBody> , next : NextFunction ) => {
49   const teacherData = {
50     id: req.params.id,
51     ...req.body,
52   };
53   try {

```

Figura 9.36: Imagen Método POST de un profesor

Una vez comprobados los datos, y verificado que son correctos, se pasará al controlador del profesor, es decir, `TeacherController.js`, ubicado en `Inteface_adapters/Controller/Teacher`. En `'TeacherController.js'`, seguirá por el método `createTeacher`.

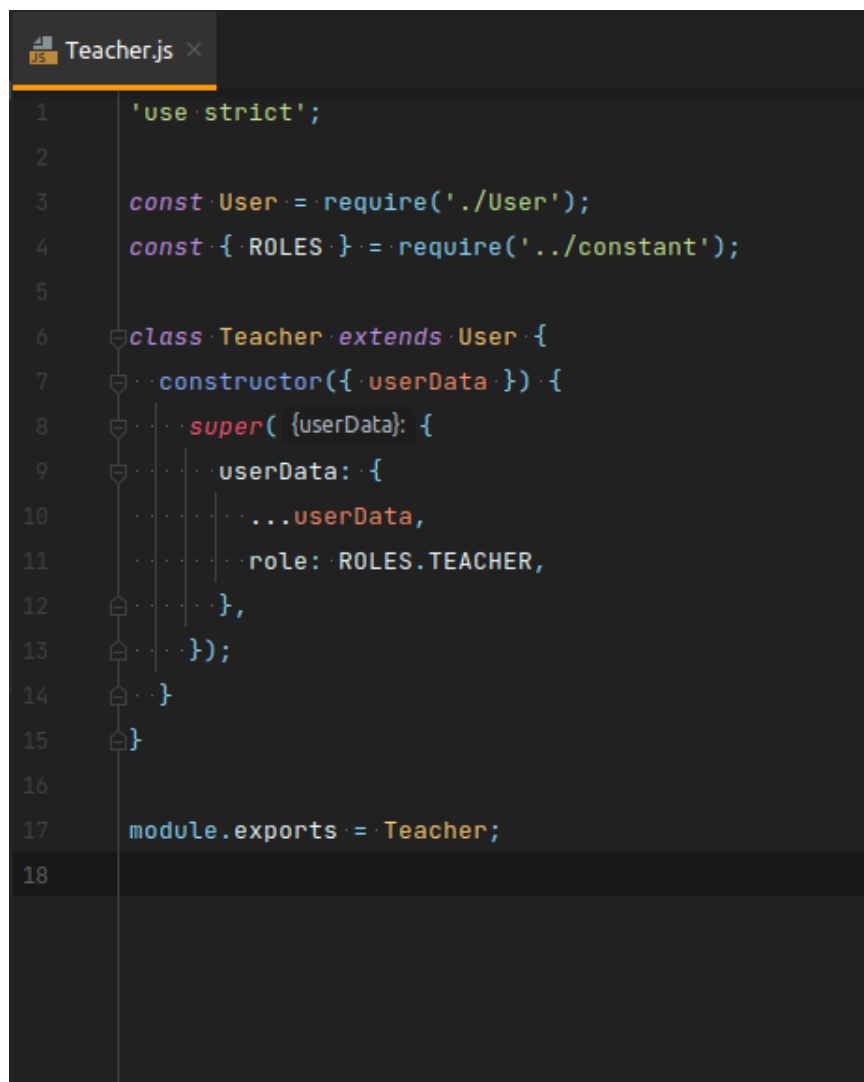


```
8
9  const UserRepository = require('../../Enterprise_business_rules/Repositories/UserRepository');
10 const UserRepositoryMySQL = require('../../Frameworks_drivers/Storage/MySQL/UserRepositoryMySQL');
11
12 const userRepositoryMySQL = new UserRepository(new UserRepositoryMySQL());
13
14 module.exports = {
15   createTeacher({ teacherData }) {
16     const teacher = new TeacherEntity({ userData: teacherData });
17     return CreateTeacher(teacher, { userRepositoryMySQL: { userRepositoryMySQL } });
18   },
19
20   async deleteTeacher(id) {
21     return DeleteTeacher(id, { userRepositoryMySQL: { userRepositoryMySQL } });
22   },
23
24   async listTeachers() {
25     return ListTeachers({ userRepositoryMySQL: { userRepositoryMySQL } });
26   },
27
28   async updateTeacher({ teacherData }) {
29     return UpdateTeacher({ teacherData: { teacherData }, { userRepositoryMySQL: { userRepositoryMySQL } });
30   },
31
32   async getTeacher(id) {
33     return GetTeacher(id, { userRepositoryMySQL: { userRepositoryMySQL } });
34   },
35 };

```

Figura 9.37: Imagen Teacher Controller

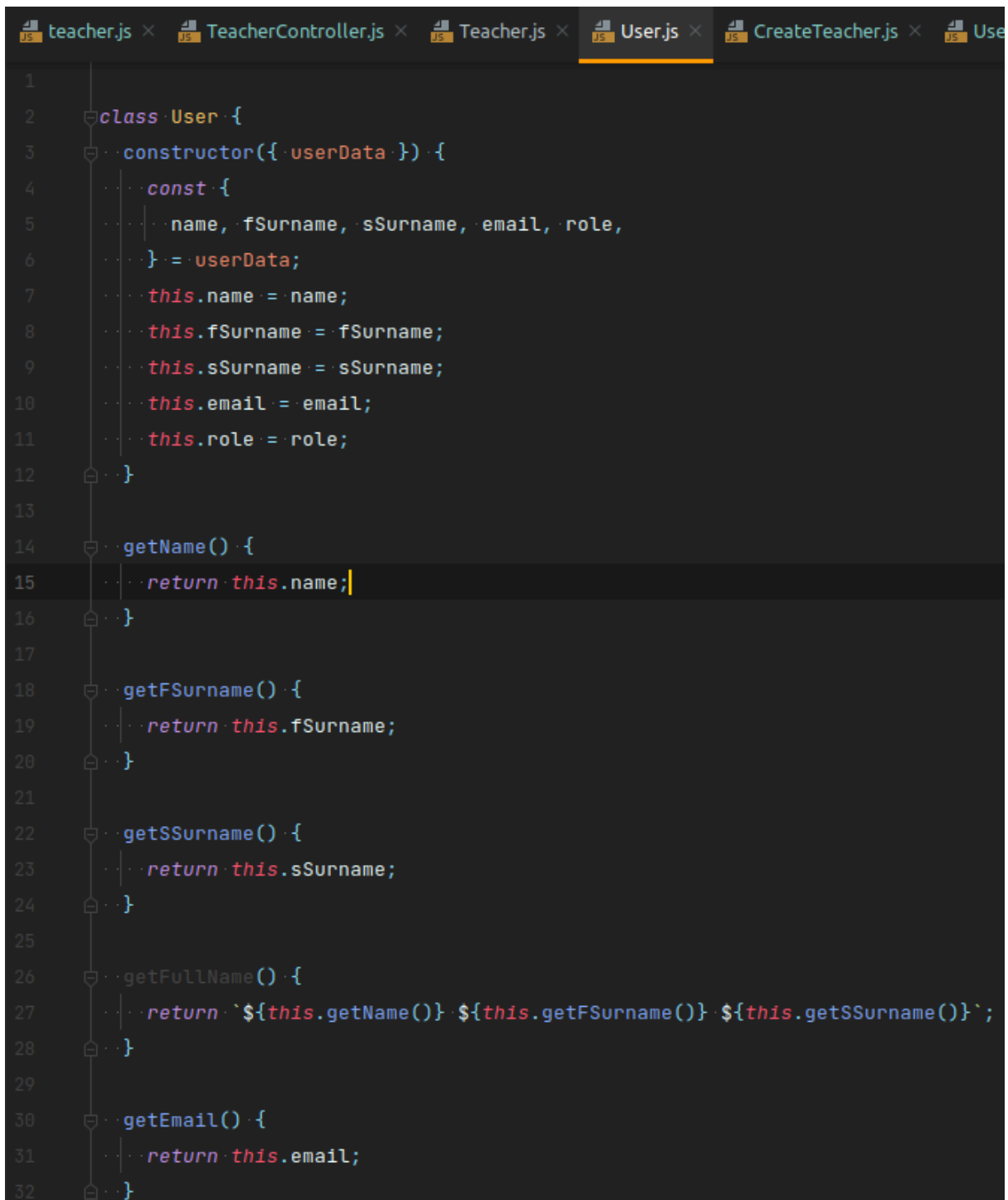
Llegado este punto, se crea una instancia de profesor utilizando los datos recibidos del enrutado. En este proceso es donde se le asigna un rol al usuario que va a ser creado, en este caso, se le va a asignar el rol de Teacher.



```
1  'use strict';
2
3  const User = require('./User');
4  const { ROLES } = require('../constant');
5
6  class Teacher extends User {
7    constructor({ userData }) {
8      super( {userData: {
9        ...userData,
10         role: ROLES.TEACHER,
11       }},
12    );
13  };
14  }
15
16
17  module.exports = Teacher;
18
```

Figura 9.38: Imagen Teacher Entity

Además del rol, está compuesta de lo que `User.js`, dado que al crear un profesor se hace una llamada a `super` y extiende de éste.

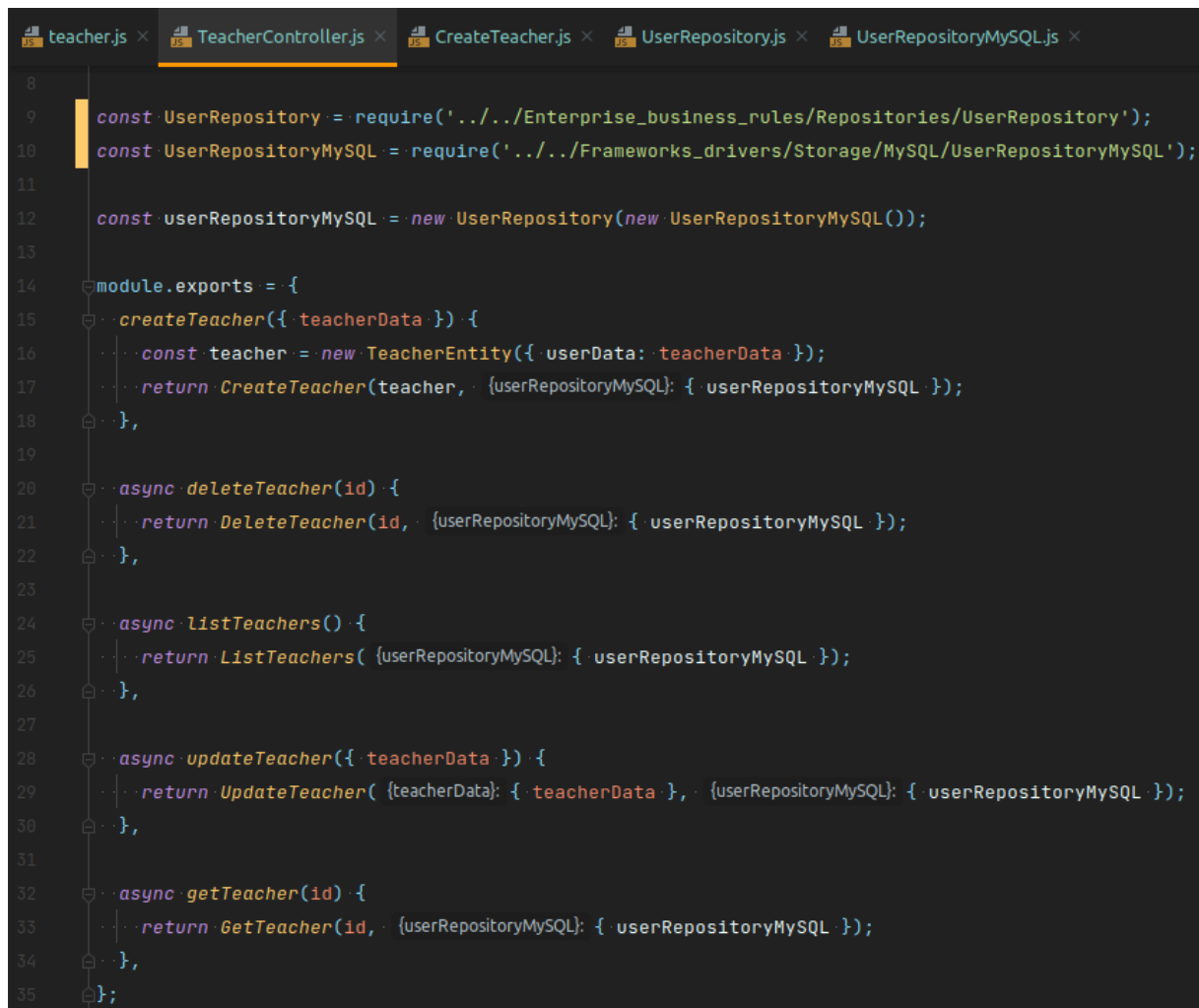


```
1
2 class User {
3   constructor({ userData }) {
4     const {
5       name, fSurname, sSurname, email, role,
6     } = userData;
7     this.name = name;
8     this.fSurname = fSurname;
9     this.sSurname = sSurname;
10    this.email = email;
11    this.role = role;
12  }
13
14  getName() {
15    return this.name;
16  }
17
18  getFSurname() {
19    return this.fSurname;
20  }
21
22  getSSurname() {
23    return this.sSurname;
24  }
25
26  getFullName() {
27    return `${this.getName()} ${this.getFSurname()} ${this.getSSurname()}`;
28  }
29
30  getEmail() {
31    return this.email;
32  }
```

Figura 9.39: Imagen Entidad User

Una vez creada la instancia del profesor, se le pasa al caso de uso junto con la instancia del repositorio.

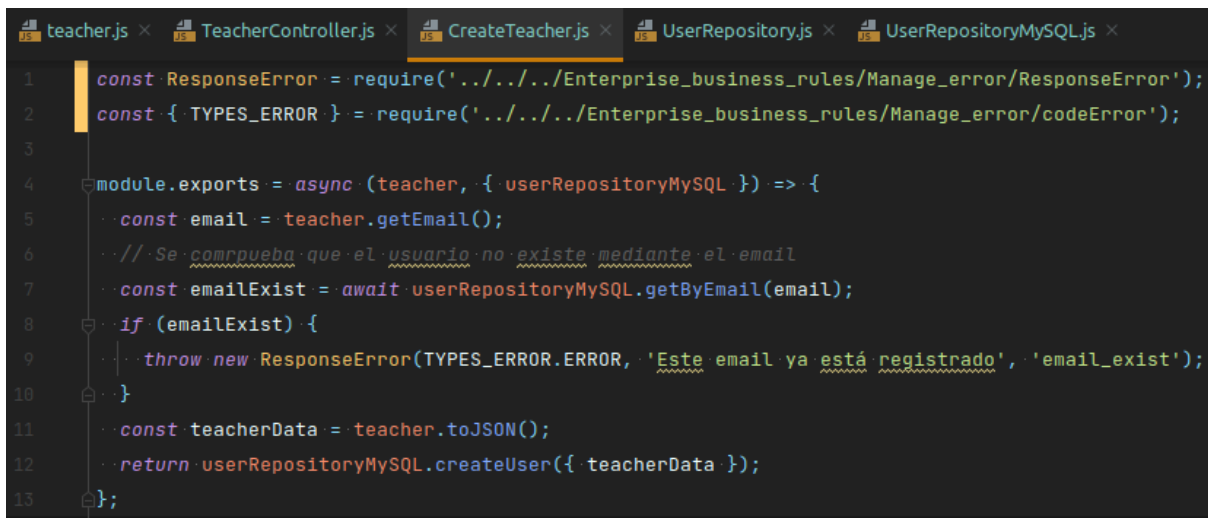




```
8
9  const UserRepository = require('../../Enterprise_business_rules/Repositories/UserRepository');
10 const UserRepositoryMySQL = require('../../Frameworks_drivers/Storage/MySQL/UserRepositoryMySQL');
11
12 const userRepositoryMySQL = new UserRepository(new UserRepositoryMySQL());
13
14 module.exports = {
15   createTeacher({ teacherData }) {
16     const teacher = new TeacherEntity({ userData: teacherData });
17     return CreateTeacher(teacher, { userRepositoryMySQL: { userRepositoryMySQL } });
18   },
19
20   async deleteTeacher(id) {
21     return DeleteTeacher(id, { userRepositoryMySQL: { userRepositoryMySQL } });
22   },
23
24   async listTeachers() {
25     return ListTeachers( {userRepositoryMySQL: { userRepositoryMySQL } });
26   },
27
28   async updateTeacher({ teacherData }) {
29     return UpdateTeacher( {teacherData: { teacherData }, { userRepositoryMySQL: { userRepositoryMySQL } });
30   },
31
32   async getTeacher(id) {
33     return GetTeacher(id, { userRepositoryMySQL: { userRepositoryMySQL } });
34   },
35 }
```

Figura 9.40: Imagen Teacher Controller

Para el caso de los casos de uso, el encargado de gestionar la creación de un profesor es `CreateTeacher.js`, ubicado en `Use_cases/Teacher/CreateTeacher.js`. Como ya se ha explicado en apartados anteriores, es aquí donde reside la lógica de la aplicación. Una vez dentro del caso de uso para crear un profesor, se analiza qué es necesario para la creación de un profesor, que el usuario no exista. Como es posible que dos personas se llamen de la misma manera, se analiza que el email que le ha sido asignado no exista, siendo ésta una manera única de identificación para un usuario.



```
1  const ResponseError = require('../../Enterprise_business_rules/Manage_error/ResponseError');
2  const { TYPES_ERROR } = require('../../Enterprise_business_rules/Manage_error/codeError');
3
4  module.exports = async (teacher, { userRepositoryMySQL }) => {
5    const email = teacher.getEmail();
6    // Se comprueba que el usuario no existe mediante el email
7    const emailExist = await userRepositoryMySQL.getByEmail(email);
8    if (emailExist) {
9      throw new ResponseError(TYPES_ERROR.ERROR, 'Este email ya está registrado', 'email_exist');
10   }
11   const teacherData = teacher.toJSON();
12   return userRepositoryMySQL.createUser({ teacherData });
13 }
```

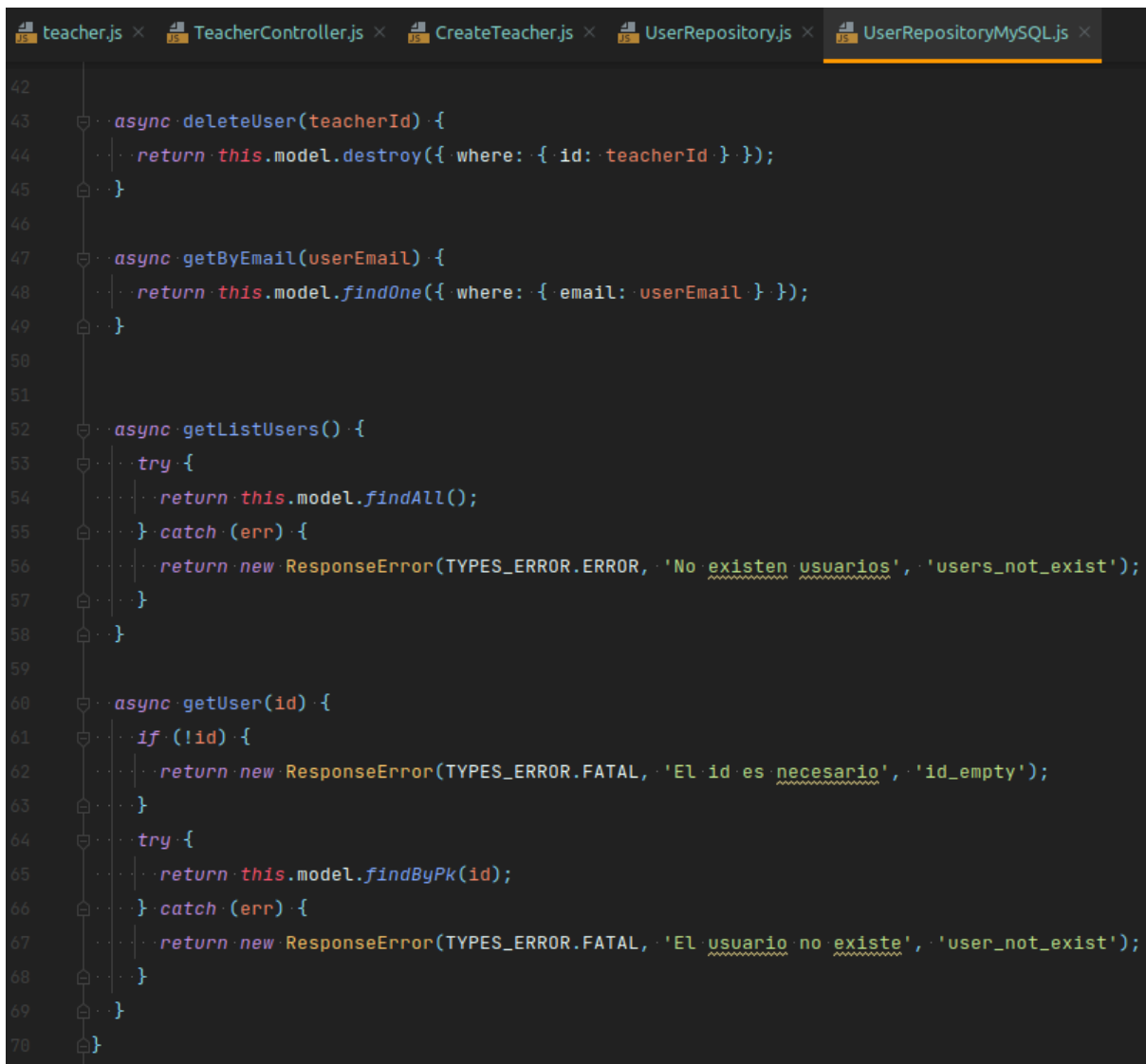
Figura 9.41: Imagen Caso de uso de la creación de un profesor

Para comprobar que el email no existe, debemos hacer una consulta a nuestra tabla de usuarios. Como no es posible acceder directamente por el principio de dependencias, utilizamos la inversión del principio de independencia haciendo uso del repositorio que hemos recibido, `UserRepositoryMySQL`, el cuál resulta ser la interfaz. Ese repositorio tiene un método llamado `getbyEmail` al que se le pasa el email recibido.

```
1  'use strict';
2
3  class UserRepository {
4    constructor(repository) {
5      this.repository = repository;
6    }
7
8    createUser({ teacherData }) {
9      return this.repository.createUser({ teacherData });
10   }
11
12   deleteUser(id) {
13     return this.repository.deleteUser(id);
14   }
15
16   getListUsers() {
17     return this.repository.getListUsers();
18   }
19
20   updateUser({ teacherData }) {
21     return this.repository.updateUser({ teacherData });
22   }
23
24   getByEmail(email) {
25     return this.repository.getByEmail(email);
26   }
27
28   getUser(id) {
29     return this.repository.getUser(id);
30   }
31 }
```

Figura 9.42: Imagen Interfaz del repositorio de usuario

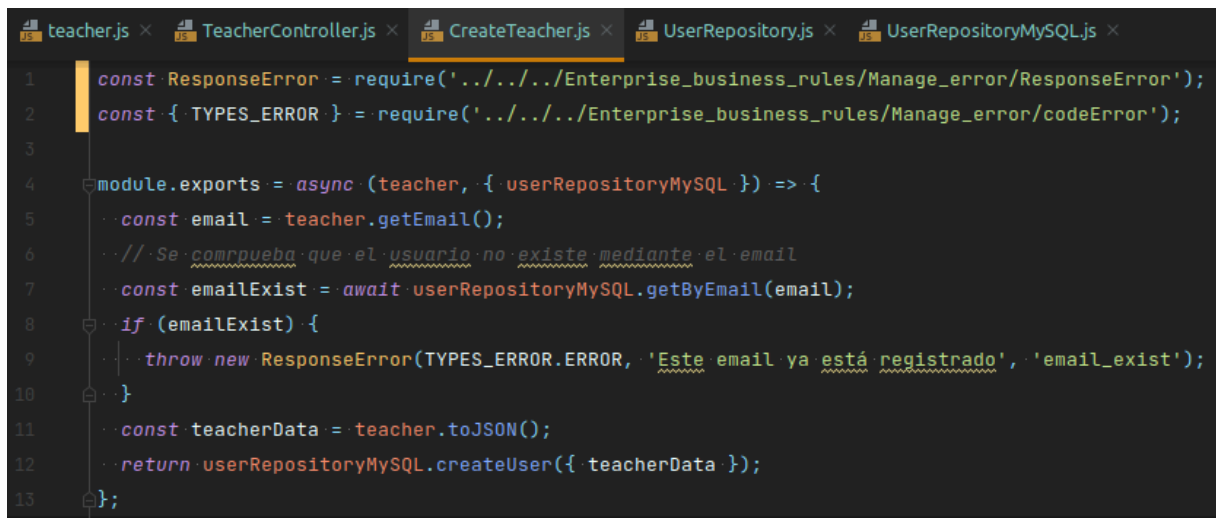
Como se aprecia en la imagen, la interfaz `UserRepository.js` dispone del método `getByEmail`, el que por herencia, redirecciona a `UserRepositoryMySQL.js`, alojado en la capa `Frameworks & Drivers` donde está implementado la interfaz `UserRepository.js`. En el método `getByEmail` de `UserRepositoryMySQL`, recibe por parámetro el email a encontrar. Para llevar a cabo la consulta, se utiliza el método `findOne` de `sequelize`, al cual se le pasa el email a buscar.



```
42
43  async deleteUser(teacherId) {
44    return this.model.destroy({ where: { id: teacherId } });
45  }
46
47  async getByEmail(userEmail) {
48    return this.model.findOne({ where: { email: userEmail } });
49  }
50
51
52  async getListUsers() {
53    try {
54      return this.model.findAll();
55    } catch (err) {
56      return new ResponseError(TYPES_ERROR.ERROR, 'No existen usuarios', 'users_not_exist');
57    }
58  }
59
60  async getUser(id) {
61    if (!id) {
62      return new ResponseError(TYPES_ERROR.FATAL, 'El id es necesario', 'id_empty');
63    }
64    try {
65      return this.model.findByPk(id);
66    } catch (err) {
67      return new ResponseError(TYPES_ERROR.FATAL, 'El usuario no existe', 'user_not_exist');
68    }
69  }
70 }
```

Figura 9.43: Imagen Implementación del repositorio de usuario

Una vez la consulta ha sido resuelta, se guarda el resultado en una constante y se analiza. Si existe un usuario con ese email, se lanzará un error. En caso negativo, los datos que se han recibido para crear el usuario serán transformados en un JSON con un método de la entidad User.



```
1  const ResponseError = require('../../Enterprise_business_rules/Manage_error/ResponseError');
2  const { TYPES_ERROR } = require('../../Enterprise_business_rules/Manage_error/codeError');
3
4  module.exports = async (teacher, { userRepositoryMySQL }) => {
5    const email = teacher.getEmail();
6    // Se comprueba que el usuario no existe mediante el email
7    const emailExist = await userRepositoryMySQL.getByEmail(email);
8    if (emailExist) {
9      throw new ResponseError(TYPES_ERROR.ERROR, 'Este email ya está registrado', 'email_exist');
10   }
11   const teacherData = teacher.toJSON();
12   return userRepositoryMySQL.createUser({ teacherData });
13 }
```

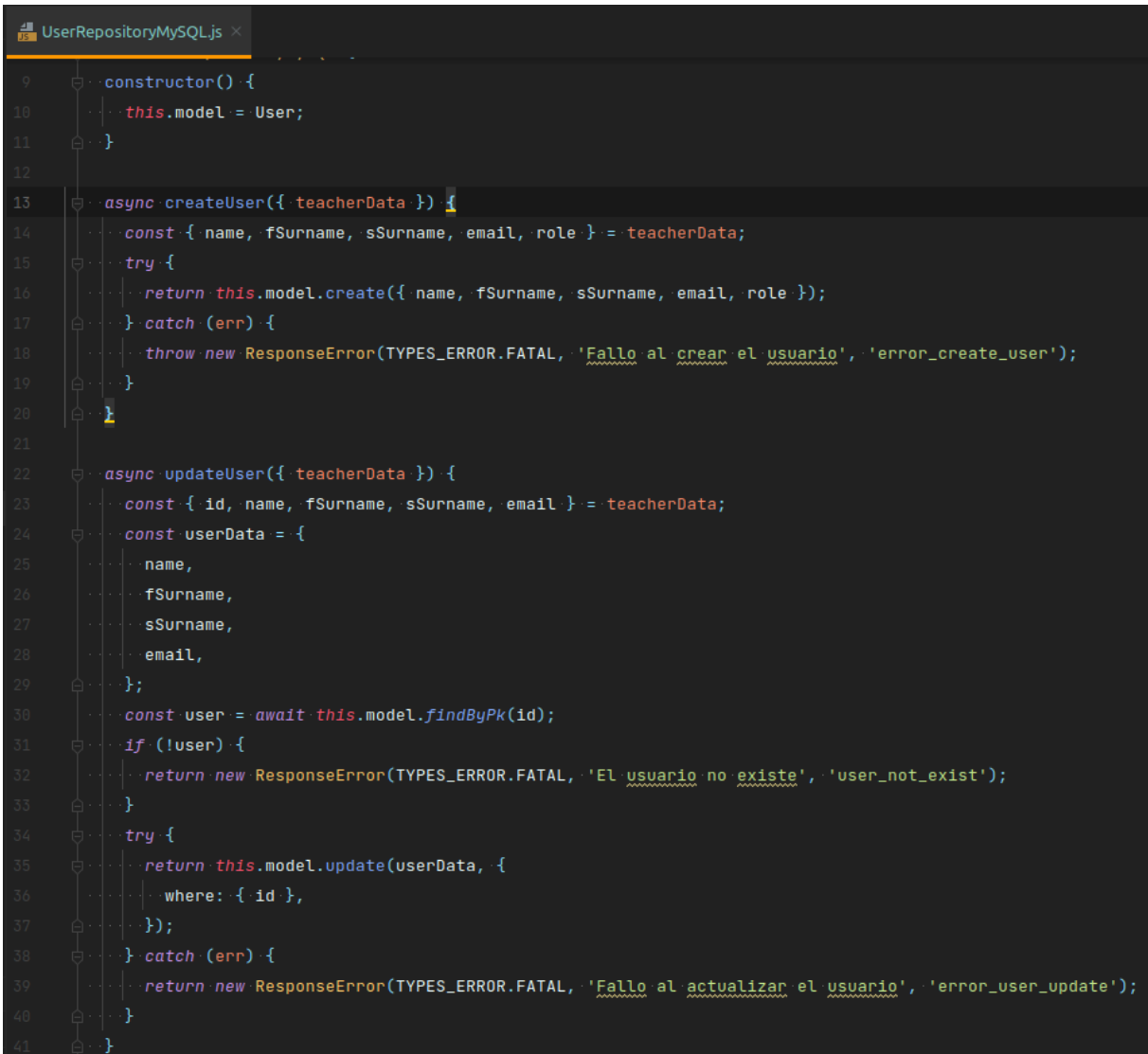
Figura 9.44: Imagen Caso de uso de la creación de un profesor

Una vez verificado que el usuario debe ser creado, los datos son enviados al repositorio, `UserRepository.js`, donde dicho repositorio tiene el método para crear usuarios, `createTeacher`.

```
1  'use strict';
2
3  class UserRepository {
4    constructor(repository) {
5      this.repository = repository;
6    }
7
8    createUser({ teacherData }) {
9      return this.repository.createUser({ teacherData });
10   }
11
12   deleteUser(id) {
13     return this.repository.deleteUser(id);
14   }
15
16   getListUsers() {
17     return this.repository.getListUsers();
18   }
19
20   updateUser({ teacherData }) {
21     return this.repository.updateUser({ teacherData });
22   }
23
24   getByEmail(email) {
25     return this.repository.getByEmail(email);
26   }
27
28   getUser(id) {
29     return this.repository.getUser(id);
30   }
31 }
```

Figura 9.45: Imagen Interfaz del repositorio de usuario

Como ya ha sido visto para el caso del email, `UserRepository.js` se trata de una interfaz que es implementada en `UserRepositoryMySQL.js`, a la cual redirige cuando recibe una petición o consulta a la base de datos. La interfaz lleva a la implementación en `UserRepositoryMySQL`, concretamente al método `createUser`, el cual recibe todos los datos requeridos para la creación del usuario y la realiza.



```
9  constructor() {
10    this.model = User;
11  }
12
13  async createUser({ teacherData }) {
14    const { name, fSurname, sSurname, email, role } = teacherData;
15    try {
16      return this.model.create({ name, fSurname, sSurname, email, role });
17    } catch (err) {
18      throw new ResponseError(TYPES_ERROR.FATAL, 'Fallo al crear el usuario', 'error_create_user');
19    }
20  }
21
22  async updateUser({ teacherData }) {
23    const { id, name, fSurname, sSurname, email } = teacherData;
24    const userData = {
25      name,
26      fSurname,
27      sSurname,
28      email,
29    };
30    const user = await this.model.findByPk(id);
31    if (!user) {
32      return new ResponseError(TYPES_ERROR.FATAL, 'El usuario no existe', 'user_not_exist');
33    }
34    try {
35      return this.model.update(userData, {
36        where: { id },
37      });
38    } catch (err) {
39      return new ResponseError(TYPES_ERROR.FATAL, 'Fallo al actualizar el usuario', 'error_user_update');
40    }
41  }
```

Figura 9.46: Imagen Implementación del repositorio de usuario

Una vez se haya realizado la consulta, se guarda en la constante `teacher`, creada en el método POST del conjunto de rutas `teacher.js`.

```

teacher.js x TeacherController.js x CreateTeacher.js x UserRepository.js x UserRepositoryMySQL.js x
22 if (Number.isNaN(Number(req.params.id))) {
23   throw new ResponseError(TYPES_ERROR.FATAL, 'El ID debe ser un número', 'id_format_error');
24 }
25 const teacher = await TeacherController.getTeacher(req.params.id);
26 res.json(teacher);
27 } catch (err) {
28   next(err);
29 }
30 });
31
32 /* Create Teacher */
33 router.post( path: '/', handlers: async (req :Request<P, ResBody, ReqBody, ReqQuery> , res :Response<ResBody> , next :NextFunction ) => {
34   try {
35     const { name, fSurname, sSurname, email } = req.body;
36     // Se comprueba si algún dato requerido no ha sido introducido
37     if (!name || !fSurname || !sSurname || !email) {
38       throw new ResponseError(TYPES_ERROR.ERROR, 'Los parámetros introducidos son incorrectos o están incompletos', 'incomplete_data');
39     }
40     const teacher = await TeacherController.createTeacher( {teacherData: req.body });
41     res.json(teacher);
42   } catch (err) {
43     next(err);
44   }
45 });
46
47 /* Modify Teacher */
48 router.put( path:('/:id)', handlers: async (req :Request<P, ResBody, ReqBody, ReqQuery> , res :Response<ResBody> , next :NextFunction ) => {
49   const teacherData = {
50     id: req.params.id,
51     ...req.body,
52   };
53   try {

```

Figura 9.47: Imagen Método POST de un profesor

Después, se envía lo que se ha recibido en formato JSON, obteniendo la siguiente respuesta en caso de que todo haya ido bien.



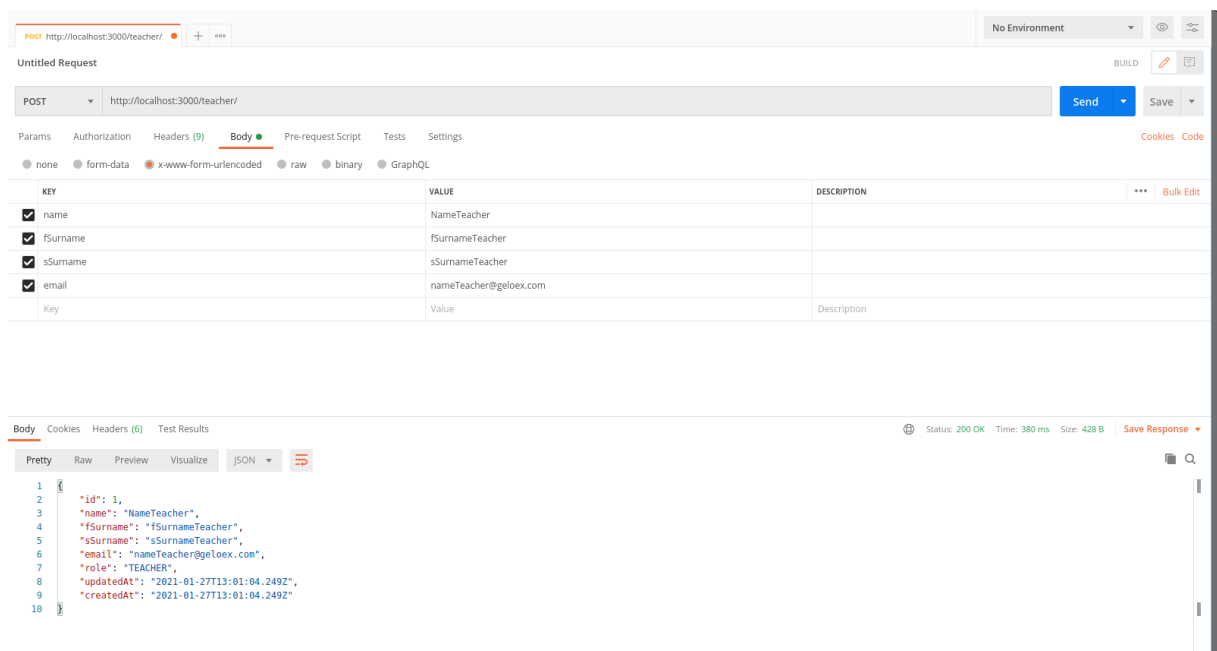


Figura 9.48: Imagen petición de creación de profesor en Postman

Por último, comprobamos que en la base de datos el profesor ha sido añadido.

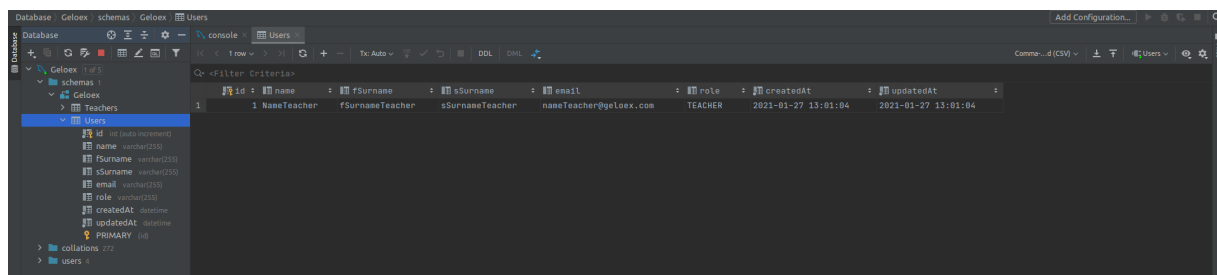


Figura 9.49: Imagen Profesor creado en la base de datos



# Capítulo 10

## Verificación y Validación

### 10.1. Introducción

Existen multitud de formas de testear software, de alcance, como pueden ser los test unitarios, los test de integración o de sistema. Los test de objetivo, como pueden ser los test de carga, los test de estrés o los test de robustex, entre otros. Tras realizar una investigación sobre para qué es necesario cada uno de ellos, se ha llegado a la conclusión de que para este proyecto los ideales son los test unitarios y los test end-to-end.

Es lógico preguntarse por qué no hacer todo tipo de test para así tener la aplicación completamente validada. La respuesta es que no es necesario. Antes de realizar la investigación, se tenía la idea de que cuantos más tipos de test se utilizaran, mejor quedaría la aplicación al final, pero esta premisa es un error. Cada test se realiza en función de la aplicación y sus necesidades, cuando la aplicación está en su primera versión, es recomendable tener cierta flexibilidad a la hora de hacer los desarrollos ya que, en cualquier momento, existe una alta probabilidad de tener que modificar gran parte de la aplicación y con ello todos los test asociados. El hecho de tener test de más al principio podría provocar retrasos en los desarrollos al tener que modificarlos o tener que eliminarlos por necesidad de rehacer la aplicación en un momento dado. Una vez la aplicación ya esté consolidada, sí sería recomendable añadir otros tipos de test en ella.

Al igual que es importante el hecho de tener validadas las funcionalidades y las funciones, también es importante la validación del código. Hoy en día, un desarrollador resulta habitual que tenga que trabajar en proyectos en conjunto con otros desarrolladores. Para que esta labor sea lo más liviana posible, las buenas prácticas de programación ayudan en ese día a día entre desarrolladores, puesto que si se realizan buenas prácticas de programación, el código resulta más legible y comprensible para el resto, ahorrando tiempo y evitando posibles errores de sintaxis.

Pese a que no se ha llegado a implementar la parte de testing, se ha realizado una especificación de la nomenclatura que se utilizaría, y los tipos de test que serían utilizados. En cambio, la parte de buenas prácticas sí ha llegado a implementarse gracias a ESLint. Además de ello, se ha utilizado la herramienta husky<sup>1</sup>. Por lo que gracias a ESLint el

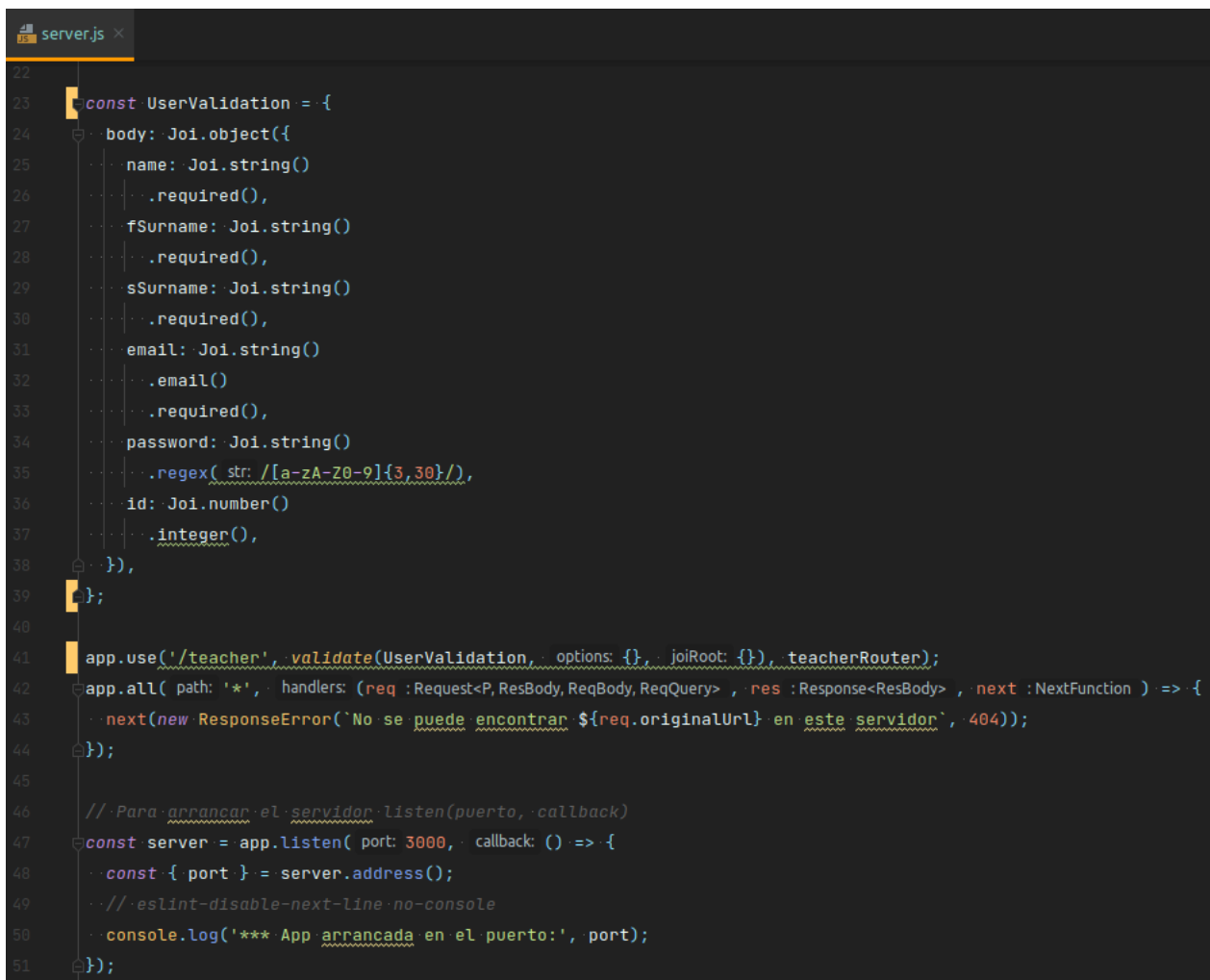
---

<sup>1</sup>Gestor de Git Hooks

código creado ha sido mediante buenas prácticas, y gracias al gestor de Git Hooks, husky, no ha sido posible realizar un commit a la rama de git si existía alguna mala práctica considerada por ESLint, ya que cuando se ejecuta el comando de commit, husky realiza un pre-commit ejecutando ESLint y buscando malas prácticas o errores de sintaxis.

## 10.2. ESLint

ESLint es una librería de análisis estático de código que se utiliza como apoyo para encontrar de una manera más rápida problemas y errores en el código, ya que cuando detecta algún tipo de ellos, te sugiere una solución, la cual si pinchas donde la sugiere, lo modifica automáticamente. También es utilizado para realizar buenas prácticas de programación y corregirlas. Además, en Webstorm, aunque no viene instalado por defecto, si que es cierto que está preparada la configuración por si se instala. A continuación, se muestra el archivo `.eslintrc.json` de este proyecto.



```
server.js
22
23 const UserValidation = {
24   body: Joi.object({
25     name: Joi.string()
26       .required(),
27     fSurname: Joi.string()
28       .required(),
29     sSurname: Joi.string()
30       .required(),
31     email: Joi.string()
32       .email()
33       .required(),
34     password: Joi.string()
35       .regex(str: /[a-zA-Z0-9]{3,30}/),
36     id: Joi.number()
37       .integer(),
38   }),
39 };
40
41 app.use('/teacher', validate(UserValidation, options: {}, joiRoot: {}), teacherRouter);
42 app.all( path: '*', handlers: (req: Request<P, ResBody, ReqBody, ReqQuery>, res: Response<ResBody>, next: NextFunction) => {
43   next(new ResponseError('No se puede encontrar ${req.originalUrl} en este servidor', 404));
44 });
45
46 // Para arrancar el servidor listen(puerto, callback)
47 const server = app.listen( port: 3000, callback: () => {
48   const { port } = server.address();
49   // eslint-disable-next-line no-console
50   console.log('*** App arrancada en el puerto:', port);
51 });
```

Figura 10.1: Imagen server

El archivo contiene el entorno que utiliza, las versiones, y las reglas. Las reglas vienen por defecto, pero es posible añadir o quitar reglas en este apartado. Como se puede apreciar, existen varias reglas que se han desactivado añadiendo la regla y poniéndola el valor en `off`.

## 10.3. Given-When-Then

Para llevar a cabo un test de cualquier tipo, será necesario realizar un análisis previo de los requisitos que debe incluir.

Para llevar a cabo esta tarea, sería recomendable seguir la nomenclatura Given-When-Then. Given-When-Then es un estilo de presentación de pruebas definido como parte del proceso de desarrollo ágil llamado *Behaviour Driven Development (BDD)*, a su vez una evolución de *Test-Driven-Development (TDD)*. Tanto en BDD como en TDD, las pruebas se ven también como una especificación (evidentemente incompleta), un enfoque que se ha llamado *Specification-by-example*. Observamos que se puede ver el parecido entre el triple "Given-When-Then" la transición de un autómata (estado-de-partida evento estado-resultante) Esta nomenclatura es una forma semiestructurada de escribir casos de prueba. A continuación, se va a explicar cada una de las partes de esta nomenclatura.

El Given será el encargado de describir el escenario, es decir, el contexto inicial del sistema. Para ello se deberá preparar al sistema y situarlo en el estado inicial. Ese punto es en que está el usuario previamente a su interacción con el sistema.

El When, por su parte, describirá la acción que se debe realizar mediante el usuario o mediante un evento emitido por otra parte del sistema.

Por último, el Then será la encargada de comprobar el resultado. Para ello comparará el resultado que se ha obtenido con el resultado esperado.

## 10.4. Test unitarios

Los test unitarios son un tipo de test que se encarga de comprobar el funcionamiento de una parte del código, es decir, de una unidad de código. Una unidad de código es cualquier porción de código pública, tal como un método público, una función pública o incluso un atributo público de una clase o componente. Para un buen testeo, deberán realizarse los test unitarios de todas las funciones, métodos o atributos públicos que tenga cada clase o componente.

Un requisito que deben cumplir nuestros test unitarios es que sean automatizables, es decir, que no sea necesaria la intervención humana para realizar el testeo. También deben ser independientes.

Esto es importante ya que en ocasiones solo queremos que se ejecute un test que nos interese para testear las modificaciones que se han llevado a cabo. Si los test no son independientes, se testearán funciones que no son necesarias y además se ralentizará el desarrollo. Además, los test unitarios deben ser completos, en el sentido que hemos definido anteriormente, ya que deben abarcar la mayor cantidad de código posible. También deben ser reutilizables. Es importante que el test no se ejecute una única vez, sino que pueda repetirse el número de veces que sea necesario. Estos test son los más ligeros, si tienen todos los requisitos que se acaban de definir.

Entrando un poco en la implementación, los pasos a seguir serían los siguientes:

1. Preparar los datos para mockear, en el caso de ser necesario.
2. Preparar las variables para los datos que se espera recibir.
3. Configurar el estado inicial de la clase o componente, para el caso en el que sea necesario testear alguna propiedad, atributo o método.
4. Ejecutar la función a testear.
5. Comprobar el resultado.

Para el caso del frontend se complica algo más, porque no hay que tener en cuenta solo los datos, que hay que testear igualmente, sino también la interfaz que se debe comprobar. Si estamos testeando el frontend, además hay que testear el DOM por si ha sufrido algún cambio.

## 10.5. Test unitarios

Los test end-to-end, al igual que los test unitarios, son un tipo de metodología encargada de testear software. La diferencia con los test unitarios es que el objetivo de los test end-to-end es testear funcionalidades completas, es decir, simular escenarios completos igual que lo haría un usuario.

Para el caso de los test end-to-end, el cómo esté hecho el código o los pormenores del sistema no serán relevantes, ya que de eso se encargan los test unitarios. Lo que sí importará será la entrada y salida que tenga el test. Por tanto, el objetivo de estos test es automatizar las pruebas de funcionalidades con el fin de tener la certeza de que todos los componentes implicados en una funcionalidad de la aplicación se ejecutan correctamente y tal y como se espera.

Esto implica que, para este caso, no está permitido el mockeo de datos, sino que debe utilizarse cada parte de la aplicación que se requiera, como la base de datos, el hardware, la red o cualquier otra aplicación que sea necesaria. Aunque, una vez se haya llevado a cabo la primera respuesta, se comprobará que el resultado obtenido es el deseado y de

ahí en adelante se podrán mockear los datos referentes a servicios externos con el fin de avanzar en el test. En este caso, es posible proceder al mockeo de datos con el fin de no bloquear el avance de la funcionalidad por si un servicio, a la hora de la prueba del test, no esté funcionando.

Estos test, debido a que prueban funcionalidades completas que implementan una interacción coherente con el usuario, son mucho más pesados de ejecutar. Por ese motivo, en una aplicación existen muchos test unitarios y pocos test end-to-end. Para llevar a cabo la implementación de un test end-to-end, se seguirá el mismo procedimiento que para los test unitarios, elaborando unos requisitos, basándonos en la nomenclatura Given-When-Then anteriormente explicada. Un test end-to-end del backend lleva a cabo la comprobación de los datos obtenidos, mientras que los test del frontend son más tediosos, ya que deben comprobar tanto los datos obtenidos como los elementos de la pantalla resultante.

Una vez en poder del Given-When-Then del test end-to-end en la que se definen los datos y los elementos que deben ser comprobados, será necesario realizar un análisis en el cual se seleccionen los elementos requeridos para la continuación del test. A partir de ahí, se empezará analizando el Given, del cual se extraen las precondiciones necesarias para darse la acción. Una vez se disponga de una buena comprensión acerca del Given, se efectuará una búsqueda en los test end-to-end realizados hasta el momento con el fin de localizar una precondición igual que la usada y de esta manera reutilizar código.

Para llevar a cabo los test, se crearán dos clases abstractas lógicas, una para el frontend y otra para el backend. Cada clase estará compuesta de dos tipos de métodos: los encargados de comprobar los datos y, en el caso del frontend, también de comprobar los elementos de la pantalla; y los programáticos, cuya finalidad es simular la acción de un usuario o actor de forma programada. Cuando debamos validar los elementos, se llevará a cabo en un primer lugar con los datos o elementos que se han seleccionado como requeridos. Una vez estos datos requeridos se hayan comprobado y validado satisfactoriamente, se proseguirá con el resto de datos y elementos del Given. Otra clase a implementar será la relacionada con la navegación. Cuando se quiera realizar la validación de una página, será necesario una navegación desde la página de inicio hasta la página a validar. Para llegar hasta ella, se deberá pasar por páginas intermedias en las cuales se deberán llevar a cabo ciertas acciones que impliquen la navegación hasta la página final. Para ello, se deben tener en cuenta todas las rutas disponibles hasta la página final, y todas deben ser incluidas. Los pasos a implementar deberán ser los más atómicos y abstractos posibles, ya que si se cambia la implementación de alguna de las páginas intermedias, la implementación de un detalle no debería afectar a la definición del paso.

Una vez creadas las clases, se procede a crear el test. En primer lugar, se lleva a cabo el Given, siendo utilizada para este punto la clase de navegación, con el objetivo de cumplir con las precondiciones antes de llevar a cabo la acción. Una vez cumplidas las precondiciones, el siguiente paso será llevar a cabo el When, en la que se realizan la acción o acciones necesarias para conseguir llegar al estado final. Una vez llegado a ese

estado final, la pantalla a validar, se procederá con el Then, llevando a cabo la validación de datos y elementos, para lo cual se utilizan los métodos de la clase abstracta lógica.



# Capítulo 11

## Trabajo futuro

El fin de este proyecto es realizar por completo una aplicación que lleve a cabo la gestión de exámenes en la Universidad. Esta aplicación debe tener dos partes claramente diferenciadas, el frontend y el backend. A pesar de que el proyecto sólo ha podido contar con una desarrolladora, y de haber tenido que rehacer la implementación desde cero, se ha logrado implementar la parte más importante del backend, quedando tan solo algunos detalles secundarios, y dejando la implementación del más sencillo frontend como trabajo futuro.

Tras realizar estudio previo a la implementación de la aplicación, indica que utilizar Clean Architecture podría ser una buena forma de organizar la parte frontend. Al igual que en la parte del backend, obtendríamos los mismos beneficios en cuanto a independencia entre capas y desacoplamiento.

Utilizando Clean Architecture, la organización de las capas sería la misma que en el backend, en lo que difiere es en el contenido, ya que en el frontend está enfocado a los datos de usuario y las vistas. La capa Enterprise Business estaría compuesta por las entidades, las cuales coinciden con las de la parte del backend. En la capa Application Bussiness, al igual que en la parte del backend, se implementaría la lógica de negocio, compuesta por los casos de uso en lo que respecta a usuarios. Para el caso de la capa Interface Adapters se llevaría a cabo la implementación de los controladores y la serialización de datos, que posteriormente, se enviará a la capa Application Bussiness. Por último, estaría la capa Framework & Drivers. En el caso de backend, esta capa alberga la implementación de la API y de la base de datos, mientras que para el caso del frontend, también se tendrá la API y se diferenciará en que en lugar de la base de datos, se tendrá la vista.

En cuanto a la arquitectura del frontend sería orientada a componentes. Esta arquitectura se basa en descomponer el diseño en componentes funcionales o lógicos. Cada componente puede ser tanto, cada una de las piezas que forman la interfaz, como la propia interfaz completa. Además, contendrá tanto la lógica como la parte visual, de este modo podemos reutilizarlo dentro de otros componentes. Como se ha anticipado brevemente, las ventajas de esta arquitectura radica en que es altamente reutilizable, que se tendrá una mínima dependencia entre componentes, no tendrán contexto específico, pudiendo así ser utilizado en diferentes ambientes y contextos, permite ser ampliado desde otro componente con el fin de crear un nuevo comportamiento y cada componente expone su interfaz para poder

ser usado por la aplicación, quedando de esta forma encapsulado.

Para llevar a cabo esta labor, las tecnologías a utilizar serían React.js y Redux, con Javascript como lenguaje. React es una librería JavaScript de código abierto creada por Facebook para desarrollar interfaces de usuarios mediante componentes. Estos componentes pueden ser tanto cada una de las piezas que forman la interfaz, como la propia interfaz completa. Cada componente contiene tanto la lógica como la parte visual, de este modo podemos reutilizarlo dentro de otros componentes. También ofrece la posibilidad de crear aplicaciones tanto en el lado del cliente como en el lado del servidor. React ofrece grandes ventajas en el desarrollo de aplicaciones web. Permite dentro de un mismo archivo con extensión `.jsx`, escribir código más legible y compacto. Al estar basado en JavaScript cuenta con una amplia comunidad con un gran número de librerías externas. Las aplicaciones webs desarrolladas con React están basadas en componentes reutilizables, esto facilita que la aplicación sea más escalable y fácil de mantener ya que los errores sucederán en la propia funcionalidad del componente o en la comunicación con los demás. Al tratarse de una librería, podemos agruparla junto a otras librerías como Redux para conseguir mayor funcionalidad y facilitar el desarrollo. Redux y React encajan muy bien ya que ambos trabajan con estados. Mientras que cada componente React tiene su propio estado (datos de cada componente que se van modificando a lo largo de su ciclo de vida), la función de Redux es emitir actualizaciones de los estados en respuesta a acciones. Y la principal ventaja de React es poder generar el DOM (“Modelo de Objetos del Documento”, estructura de los elementos que se generan en el navegador web al cargar una página) de forma dinámica. Esto permite que para poder visualizar los cambios de los datos, no es necesario renderizar toda la página de nuevo, sino solamente el componente que haya sido actualizado. Gracias a esta característica mejora: la experiencia de usuario al navegar por la aplicación web, la rapidez en la carga de las páginas y facilita el mantenimiento de la aplicación.

Redux es una librería JavaScript que ayuda a la gestión del estado de la aplicación y el estado de cada uno de los componentes. Funciona emitiendo actualizaciones de estado en respuesta a acciones, con la peculiaridad de realizar dichas modificaciones a través de objetos sencillos llamados **actions**, y no a través de cambios directos en el estado. Los principios de Redux son:

- Existe un objeto Store en el cual se almacena el estado, en formato JSON, de toda la aplicación en un árbol.
- El estado será solo de lectura, solo será posible modificarlo emitiendo una acción (action) con un objeto que describe lo ocurrido.
- Los cambios se llevarán a cabo en funciones puras llamadas reducers, en ellas se generará un nuevo estado sin modificar el anterior.

Las ventajas de la utilización de Redux son amplias. Al desacoplar la gestión del estado de los componentes y trasladarlo a un único sitio, damos robustez a nuestra aplicación, evitando que cuando nuestra aplicación crezca sea complejo manejar el estado y donde ubicarlo. Se facilita la comunicación entre componentes ya que es posible hacerlo de forma

sencilla mediante el flujo de acciones y sin necesidad de acoplar un componente a otro. En el desarrollo web resulta costoso consultar el DOM, por lo que no es recomendable respaldarnos en el DOM para manejar el estado de nuestra aplicación, en cambio Redux obtiene la información y la modifica con un coste muy bajo, esto implica que gracias a Redux, obtendremos una mejora del rendimiento en lo que a gestión del estado se refiere. Una de las principales ventajas a la hora de implantar Redux en nuestra aplicación es que es independiente del lenguaje o framework que estemos usando, es posible utilizarlo con cualquier lenguaje de programación. Resulta ser muy eficiente la forma en la que gestiona y pone a disposición los datos que son utilizados en las vistas, llegando incluso a disponer de los datos en tiempo real.



# Capítulo 12

## Conclusiones

Éste proyecto trata de la continuación de un TFG[34] de hace dos años, el cual surgió de la necesidad de mejorar la gestión de exámenes de la Universidad Complutense. En el TFG de los compañeros, realizaron una gran especificación y análisis de la aplicación, pero lamentablemente no llegaron a implementar la mayor parte y lo que estaba implementado era estático o estaba mal, desde el punto de vista de su especificación. Es por ello que se ha continuado, para conseguir llevar a la práctica esa especificación.

Además de basarse en la especificación de los compañeros, también se han realizado mejoras y han sido añadidas funcionalidades que no habían considerado. En este proyecto, se ha logrado crear una estructura backend basada en Clean Architecture[30][29] de Robert C.Martin, lo que proporciona una estructura válida para cualquier proyecto sin necesidad de ser de logística de exámenes. El backend se ha dividido en las cuatro capas de Clean Architecture, Enterprise Business Rules, Application Business Rules, Interface Adapters y Frameworks & Drivers. Dentro del proyecto, la capa Enterprise Business Rules es la capa encargada de alojar las entidades del proyecto con las funciones más básicas de la aplicación. Además, contará con un repositorio de interfaces que conectarán la parte de la lógica de la aplicación con la base de datos, creando de esta forma la inversión de dependencias[39] necesaria para cumplir con la regla de dependencias de Clean Architecture y el patrón DAO. La capa Application Business Rules por su parte, es la encargada del desarrollo de la lógica de negocio, albergando aquí los casos de uso que desarrolla la aplicación. La siguiente capa es la de Interface Adapters, esta capa se encarga de la transformación de datos, de establecer los controladores necesarios y del enrutado de toda la aplicación. Por último, Frameworks & Drivers, es la capa más dependiente y con más posibilidad de cambios, ya sean por servicios de terceros o por la implementación de la base de datos, la cual se aloja en esta capa. Suele pasar que el cambio del tipo de base de datos en una aplicación genere muchos problemas, es por ello que se ha utilizado el ORM Sequelize para en el caso de necesitar realizar un cambio en la base de datos, solo sería necesario cambiar el tipo en la configuración de Sequelize. Es importante mencionar que la implementación de la base de datos está basada en la interfaz del repositorio de la capa Enterprise Business Rules, por lo tanto, todo método que se encuentre en esa interfaz, deberá estar implementada en la capa Frameworks & Drivers.

Una vez realizado el proyecto, y tras hacer un análisis de ello, se puede considerar que

es mucho lo que se ha aprendido. Es muy importante basar el proyecto en una buena arquitectura, como es caso de Clean Architecture, para obtener una gran escalabilidad, aunque al principio sea complicado de entender o haya que rehacer todo porque se haya roto la regla de dependencias por todas partes. Otro factor importante ha sido el de aprendizaje de la asincronía de Javascript, y el ser consciente de que se ejecuta en un hilo y es necesario manejar esta asincronía con el fin de no bloquear la aplicación. También se ha aprendido la importancia de los mediadores, como pueden ser los controladores, ORM o DAO, independizando la base de datos y el enrutado de la lógica de la aplicación. Y por último, de lo útil que es tener una gestión de errores ya que, es mucho más rápido encontrar los errores.

# Capítulo 13

## Conclusion

This project is about the continuation of a TFG [34] two years ago, which arose from the need to improve the management of exams at the Complutense University. In the TFG of the colleagues, they carried out a great specification and analysis of the application, but unfortunately they did not implement most of it and what was implemented was static or was wrong, from the point of view of its specification. That is why it has been continued, in order to put that specification into practice.

In addition to being based on the specification of the companions, improvements have also been made and functionalities have been added that they had not considered. In this project, it has been possible to create a backend structure based on Clean Architecture [30][29] by Robert C. Martin, which provides a valid structure for any project without the need for exam logistics. The backend has been divided into the four layers of Clean Architecture, Enterprise Business Rules, Application Business Rules, Interface Adapters, and Frameworks & Drivers. Within the project, the Enterprise Business Rules layer is the layer in charge of hosting the project entities with the most basic functions of the application. In addition, it will have a repository of interfaces that will connect the part of the application logic with the database, thus creating the dependency inversion [39] necessary to comply with the Clean Architecture dependency rule and the DAO pattern. The Application Business Rules layer, for its part, is in charge of developing the business logic, hosting here the use cases developed by the application. The next layer is Interface Adapters, this layer is in charge of data transformation, establishing the necessary drivers and routing the entire application. Finally, Frameworks & Drivers, is the layer that is most dependent and with the greatest possibility of changes, either by third-party services or by the implementation of the database, which is housed in this layer. It usually happens that changing the database type in an application generates many problems, that is why the Sequelize ORM has been used to in the case of needing to make a change in the database, it would only be necessary to change the type in the Sequelize configuration. It is important to mention that the implementation of the database is based on the interface of the repository of the Enterprise Business Rules layer, therefore, any method found in that interface must be implemented in the Frameworks & Drivers layer.

Once the project is completed, and after doing an analysis of it, it can be considered that much has been learned. It is very important to base the project on a good architecture,

as in the case of Clean Architecture, to obtain great scalability, even if at the beginning it is difficult to understand or you have to redo everything because the dependency rule has been broken everywhere. Another important factor has been learning Javascript asynchrony, and being aware that it is running in a thread and it is necessary to handle this asynchrony in order not to block the application. The importance of mediators, such as controllers, ORM or DAO, has also been learned, making the database and routing independent of the application logic. And finally, how useful it is to have an error management since it is much faster to find the errors.



# Bibliografía

- [1] Comparativa entre express, hapi y koa. <https://medium.com/@theomalaper.cognez/express-vs-koa-and-hapi-a2c65f949b78>, ????
- [2] Definición de orm. <https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html>, ????
- [3] Express. <https://expressjs.com/es/>, ????
- [4] If i were oracle, i would be pretty scared because the reason for my existence is evaporating from underneath me.[...] the reason for the database to exist is disappearing. [https://www.youtube.com/watch?t=3157&v=o\\_TH-Y78tt4&feature=youtu.be](https://www.youtube.com/watch?t=3157&v=o_TH-Y78tt4&feature=youtu.be), ????
- [5] Información acerca de promesas. [https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Usar\\_promesas](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Usar_promesas), ????
- [6] Información sobre deno. <https://blog.bitsrc.io/what-is-deno-and-will-it-replace-nodejs-a13aa1734a74>, ????
- [7] Información sobre node. <https://medium.com/@sebastianpaduano/welcome-to-node-js-peque%C3%B1a-introducci%C3%B3n-163d0299de81#:~:text=js%20es%20un%20entorno%20en,el%20motor%20V8%20de%20Google.>, ????
- [8] Instalación de mysql. <https://medium.com/@shivraj.jadhav82/mysql-setup-on-linux-mint-948470115d5>, ????
- [9] Mención robert c.martin. [https://www.youtube.com/watch?v=o\\_TH-Y78tt4&feature=youtu.be&t=2771](https://www.youtube.com/watch?v=o_TH-Y78tt4&feature=youtu.be&t=2771), ????
- [10] Página oficial sequelize. <https://www.npmjs.com/package/sequelize>, ????
- [11] There will probably be some relational tables that survive, but now there is some healthy competition. [https://www.youtube.com/watch?t=3216&v=o\\_TH-Y78tt4&feature=youtu.be](https://www.youtube.com/watch?t=3216&v=o_TH-Y78tt4&feature=youtu.be), ????
- [12] Manejo de errores con express. <https://expressjs.com/es/guide/error-handling.html>, 2017.
- [13] Clean architecture. <https://www2.deloitte.com/es/es/pages/technology/articles/clean-architecture.html>, 2019.

- [14] Información acerca de promesas. <https://dev.to/lydiahallie/javascript-visualized-promises-async-await-5gke>, 2019.
- [15] Manejo de errores con express. <https://zellwk.com/blog/async-await-express/>, 2019.
- [16] express-validation. <https://www.npmjs.com/package/express-validation>, 2020.
- [17] Información acerca de express. [https://developer.mozilla.org/es/docs/Learn/Server-side/Express\\_Nodejs/Introduction/](https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction/), 2020.
- [18] Información acerca de express. [https://developer.mozilla.org/es/docs/Learn/Server-side/Express\\_Nodejs/Introduction/](https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction/), 2020.
- [19] Instalación de node.js. <https://github.com/nodesource/distributions/blob/master/README.md#rpm>, 2020.
- [20] Intalación sequelize. <https://www.npmjs.com/package/sequelize>, 2020.
- [21] Tutorial sequelize parte 1. <https://www.youtube.com/watch?v=T6rGUZGAWBk>, 2020.
- [22] Tutorial sequelize parte 2. <https://www.youtube.com/watch?v=0bl0vfV3g-c>, 2020.
- [23] Tutorial sequelize parte 3. <https://www.youtube.com/watch?v=0bl0vfV3g-c>, 2020.
- [24] Tutorial sequelize parte 4. <https://www.youtube.com/watch?v=50dugGTFvi8>, 2020.
- [25] Ventajas de node. <https://www.simform.com/nodejs-advantages-disadvantages/>, 2020.
- [26] Ventajas de node.js. <https://codewithhugo.com/node-pros-and-cons/>, 2020.
- [27] BENÍTEZ, C. Test unitarios. <https://www.etnassoft.com/2011/07/27/tests-unitarios-cuando-usarlos-y-pistas-para-conseguir-un-sistema-robusto/>, 2011.
- [28] CM, S. Ventajas de express. <https://www.techomoro.com/what-are-the-benefits-of-using-express-js-for-backend-development/>, 2019.
- [29] C.MARTIN, R. Clean architecture. <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>, 2012.
- [30] C.MARTIN, R. *Clean Architecture*. Prentice-Hall, 2017.
- [31] COCKBURN, A. Hexagonal architecture. <https://alistair.cockburn.us/hexagonal-architecture/>, ????

- [32] DEUTSCH, D. Clean architecture. <https://www.freecodecamp.org/news/a-quick-introduction-to-clean-architecture-990c014448d2/>, 2018.
- [33] FEOKTISTOV, I. Ventajas de node.js. <https://relevant.software/blog/7-benefits-of-node-js-for-startups/>, 2020.
- [34] MARÍN, I. D., REYES, J. F. y ANDREU, Á. S. *Gestión de la logística de exámenes en la Universidad*. Tesis Doctoral, Universidad Complutense de Madrid, 2019.
- [35] MAURICIO, A. Arquitectura hexagonal. [https://our-academy.org/posts/arquitectura-hexagonal, ????](https://our-academy.org/posts/arquitectura-hexagonal,????)
- [36] MEYER, B. *Object Oriented Software Construction*. Prentice-Hall, 1988.
- [37] MORENO, O. Test unitarios. <http://oscarmoreno.com/pruebas-unitarias/>, 2019.
- [38] OLIVEIRA, D. Clean architecture. <https://www.freecodecamp.org/news/how-to-write-robust-apps-consistently-with-the-clean-architecture-9bdca93e17b/>, 2017.
- [39] SÁNCHEZ FERNÁNDEZ, J. Clean architecture. <https://xurxodev.com/profundizando-en-la-inversion-de-dependencia/>, 2016.
- [40] SÁNCHEZ, J. Clean architecture. <https://xurxodev.com/por-que-utilizo-clean-architecture-en-mis-proyectos/>, 2016.
- [41] WAGENER, D. Manejo de errores con express. <https://medium.com/@SigniorGratiano/express-error-handling-674bfdd86139>, 2019.
- [42] WHITE, L. Ventajas de express. <https://codersera.com/blog/learn-express-js/>, 2020.
- [43] WIKIPEDIA (LaTeX). Entrada: “Definición ad hoc”. Disponible en [https://es.wikipedia.org/wiki/Ad\\_hoc](https://es.wikipedia.org/wiki/Ad_hoc) (último acceso, Enero, 2021).
- [44] WIKIPEDIA (LaTeX). Entrada: “Definición DAO”. Disponible en [https://es.wikipedia.org/wiki/Objeto\\_de\\_acceso\\_a\\_datos](https://es.wikipedia.org/wiki/Objeto_de_acceso_a_datos) (último acceso, Julio, 2019).
- [45] WIKIPEDIA (LaTeX). Entrada: “Definición Domain-Driven Design”. Disponible en [https://es.wikipedia.org/wiki/Dise%C3%B1o\\_guiado\\_por\\_el\\_dominio](https://es.wikipedia.org/wiki/Dise%C3%B1o_guiado_por_el_dominio) (último acceso, Julio, 2019).
- [46] WIKIPEDIA (LaTeX). Entrada: “Información acerca de Node.js”. Disponible en <https://es.wikipedia.org/wiki/Node.js> (último acceso, Mayo, 2014).
- [47] WIKIPEDIA (LaTeX). Entrada: “Principio de sustitución de liskov”. Disponible en [https://es.wikipedia.org/wiki/Principio\\_de\\_sustituci%C3%B3n\\_de\\_Liskov](https://es.wikipedia.org/wiki/Principio_de_sustituci%C3%B3n_de_Liskov) (último acceso, Septiembre, 2019).
- [48] WIKIPEDIA (LaTeX). Entrada: “Prueba unitaria”. Disponible en [https://es.wikipedia.org/wiki/Prueba\\_unitaria](https://es.wikipedia.org/wiki/Prueba_unitaria) (último acceso, Mayo, 2014).



